Advanced search

Linux Journal Issue #114/October 2003

Archive Index

Advanced search

Bootable Restoration CDs with Mondo

Craig Swanson

Matt Lung

Issue #114, October 2003

A quick way to restore from bare metal or clone systems is to build a custom rescue CD image. Here's how to automate the process and keep the rescue CDs up to date.

The disaster recovery suite Mondo has become a vital component of the backup and restore plan for Midwest Tool & Die (MTD). With the addition of Mondo's backup tool, mondoarchive, to our backup plan, we gained disaster recovery from bare metal, quick rollback to known good configurations and easy duplication of Linux software loads. Mondo also has the ability to support the cloning of LVM, RAID, ext2, ext3, JFS, XFS, ReiserFS and VFAT filesystems.

Disaster Recovery

At MTD, we rely on tape to back up company and user data. We have used several enterprise backup solutions, but disaster recovery has been an issue with each one. In the past, to recover from bare metal required re-installation of Linux from the distribution CDs. After the fresh installation, the tape backup client software was loaded. Finally, the system's specific software was restored from the tape backup.

Now, we create a bootable restoration CD that can recover an entire system with its specific drivers and application software. A tape restore then overwrites files that have changed since the CD was created. Whenever significant system changes have been made, a new rescue CD image is created to replace the existing copy.

Regular System Backup

Creating a rescue CD image is a straightforward process, so it can be repeated whenever a snapshot of a system is needed. With today's large hard disks, periodic backups are possible with mondoarchive. A cron job automation example to handle this is explained later in this article.

Here, at MTD, we use mondoarchive instead of tape to back up certain systems:

- For Linux machines that serve static data, tape backup is not necessary. These machines can be backed up periodically with mondoarchive.
- Sacrificial systems exposed to the outside Internet through a demilitarized zone. These systems must be segregated from the internal network and are not visible to the tape backup server. mondoarchive can maintain a good copy of sacrificial systems. If a system's security is compromised, restoration is quick.
- Linux firewall routers having static configurations with no user accounts and high security. For this reason, routers are not accessible by our enterprise tape backup system. The router configurations change infrequently, so mondoarchive is a good fit.

Other Applications

In addition to backups, a rescue CD can serve as a bootable, restorable system snapshot. mondoarchive's ease of use has spawned several applications at our company.

Building a test bed is another easy task with mondoarchive. A system can be loaded from bare metal and the known good load can be restored at will. When the final system load is ready for production, a new CD image is created. This becomes the disaster recovery and quick rollback CD for the production server.

Mondo's restore utility is flexible. With the interactive restore utility, we resize and restructure partitions. This is a useful method for upgrading from an existing hard disk to a larger disk.

To duplicate an installed Linux system, create a rescue CD from the entire system. Then, restore it to another bare metal box. If the original system was configured to use DHCP for IP address assignment, the new system will do the same. For systems with static IPs, both systems now have the same hostname and IP address. So, make sure to load the clone while disconnected from your production network. We use a test bed network for this purpose.

As if these uses weren't enough, mondoarchive also includes an option to verify the archived files with your system; therefore, you can use the CD as a benchmark against the present system. This is useful for checking the integrity of static systems.

As you can see, mondoarchive is a flexible and useful archive and cloning utility. If you value your data and/or server configuration, read on.

Dependencies

The installation examples here pertain to Red Hat Linux 8.0, which is our tested environment. Several dependencies need to be satisfied before you can actually install mondoarchive. On Red Hat systems, check for these required packages: afio, cdrecord, buffer, mkisofs, syslinux and bzip2. If they all are installed, you can move on to installing the mondoarchive packages. If not, you have a little bit of prep work to do. In most cases you can find these packages on your Red Hat CDs. Alternatively, you could download them from Red Hat, rpmfind.net, or directly from the Mondo home page. When you have located all the packages, install them with `rpm`.

After you have satisfied the dependencies, you can install the packages themselves. Two packages specifically need to be installed for mondoarchive. The mindi and mondo packages can be downloaded from www.microwerks.net/~hugo/index.html.

mindi is the portion of Mondo that creates boot and root floppies/CDs. It basically makes sure that mondoarchive has everything it needs to boot the archival CD or floppies. Install mindi first with:

```
# rpm -Uvh mindi
```

You also have to install the mondo package. After mindi is installed, type:

```
# rpm -Uvh mondo
```

Execution

Many different types of systems can be backed up in various ways with mondoarchive. Here we describe only the situations we laid out earlier—to back up our servers and create clone systems.

In our environment, many servers perform various tasks, and each of them is configured differently. Some have multiple IDE or SCSI hard disks for massive storage, and some have only one IDE or SCSI drive. There even are some RAID systems. On some servers the data is ever-changing, and on some servers it hardly ever changes. It is possible to use mondoarchive to clone all of these systems.

First of all, it may be a good idea to look at your disk usage on a per-server basis. Pay close attention to what is being mounted, where and when. There is no need to back up noncritical information if it can be avoided; if you have large directories that do not contain critical data, you should think about excluding them. For example, we share data between servers over NFS and automount, and we have many of the same shares mounted on each server. What you don't want to do is ignore these mounted shares and have mondoarchive back up that data too. After you have identified the unnecessary mounted partitions or shares, you have the ability to exclude them with the -E option. The format of that option should be as follows: `-E /a /b /c` where /a /b and /c are directories. This will ensure exclusion of that data.

Storing and Burning the ISO Image

Now that you know exactly what you want to back up, let's examine the mondoarchive command and a few of its options. You have the ability to back up to CD, ISO images and an NFS share. In this article we discuss only how to back up to ISO images for burning to CD at a later time. For complete details on mondoarchive and its usage, read its man page.

Before you run the mondoarchive command, choose a place on your drive that has a lot of room to store a large ISO image file. Say we pick /home/mondo, and home is a 6GB partition. The command to use looks like this:

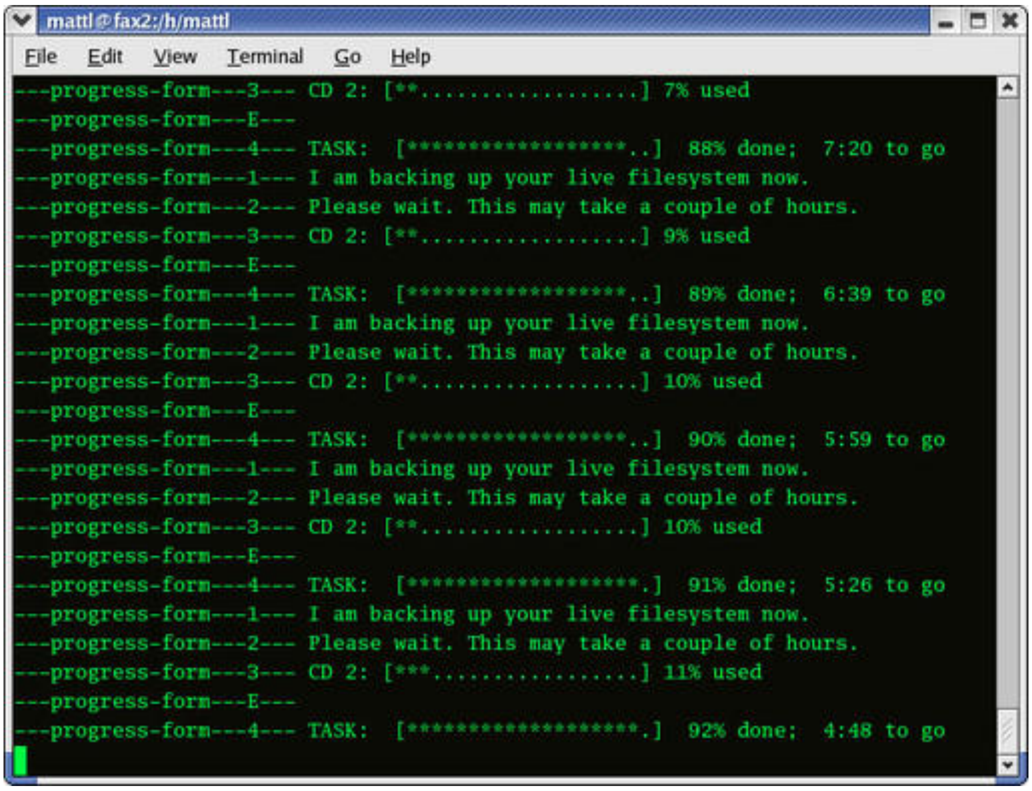
```
# mondoarchive -Oi -d /home/mondo -E "/home/mondo"
```

The -Oi option tells mondoarchive to back up the filesystem to an ISO image. Next, `-d /home/mondo` lets mondoarchive know the resulting ISO images should be put in the /home/mondo directory. Depending on the size of your system, you may have multiple ISO images created. Finally, `-E /home/mondo` excludes unnecessary directories. Here, we told it to exclude /home/mondo, which could contain other massive ISO images and cause your backup to grow unnecessarily.

In cases where disk space is low, you need to specify a scratch directory. This is a temporary directory that mondoarchive uses to build its ISO images before they are archived. In this situation, it is wise to tell mondoarchive to put its scratch directory in a large partition. Otherwise, mondoarchive most likely will fail when it runs out of room. In the example below, pretend /var/local/data is a large partition on your disk. To specify the scratch directory, run the mondoarchive command adding an -S option:

```
# mondoarchive -Oi -d /home/mondo \
-S /var/local/data -E "/home/mondo"
```

After running the command, mondoarchive checks your system, makes sure everything is okay and begins its backup process. It continuously shows its progress during the process (Figure 1) and may take a while to complete. When it finally is finished, it asks if you want to create boot disks. You can answer no, because the CD you burn will be bootable. If you want or need the disks, say yes.



```
---progress-form---3--- CD 2: [**.................] 7% used
---progress-form---E---
---progress-form---4--- TASK:  [******************..]  88% done;  7:20 to go
---progress-form---1--- I am backing up your live filesystem now.
---progress-form---2--- Please wait. This may take a couple of hours.
---progress-form---3--- CD 2: [**.................] 9% used
---progress-form---E---
---progress-form---4--- TASK:  [******************..]  89% done;  6:39 to go
---progress-form---1--- I am backing up your live filesystem now.
---progress-form---2--- Please wait. This may take a couple of hours.
---progress-form---3--- CD 2: [**.................] 10% used
---progress-form---E---
---progress-form---4--- TASK:  [******************..]  90% done;  5:59 to go
---progress-form---1--- I am backing up your live filesystem now.
---progress-form---2--- Please wait. This may take a couple of hours.
---progress-form---3--- CD 2: [**.................] 10% used
---progress-form---E---
---progress-form---4--- TASK:  [*******************.]  91% done;  5:26 to go
---progress-form---1--- I am backing up your live filesystem now.
---progress-form---2--- Please wait. This may take a couple of hours.
---progress-form---3--- CD 2: [***................] 11% used
---progress-form---E---
---progress-form---4--- TASK:  [*******************.]  92% done;  4:48 to go
```

Figure 1. Mondoarchive Running

When it's complete, you'll have ISO images in /home/mondo or wherever you specified, from which you can burn CDs. You can burn them in many different ways, including using Xcdroast, Webmin or the cdrecord command. To do this quickly, run `cdrecord -scanbus`. This discovers your CD writer's bus, target and logical unit number (LUN), which is usually 0,0,0:

```
# cdrecord dev=0,0,0 speed=xx /home/mondo/1.iso
```

When cdrecord is done, you have a restore disk for your server if it goes down.

Running as a Cron Job

mondoarchive also can be run automatically at a time of your choosing by setting it up as a cron job. To set this up, first create a script similar to the following and place it in /etc/cron.daily/:

```
#!/bin/sh
mkdir -p /home/mondo/`date +%A` && \
mondoarchive -Oi -d /home/mondo/`date +%A` \
-E /home/mondo
```

When placed in /etc/cron.daily/, this script runs every day at the same time. Upon execution, it creates a folder in /home/mondo corresponding to the day. If you run the cron job seven days a week, there will be seven folders in /home/mondo, each named for a day of the week and containing the ISO images for that day's backup. Of course, if you want to have these on CD, you can use the cdrecord command again.

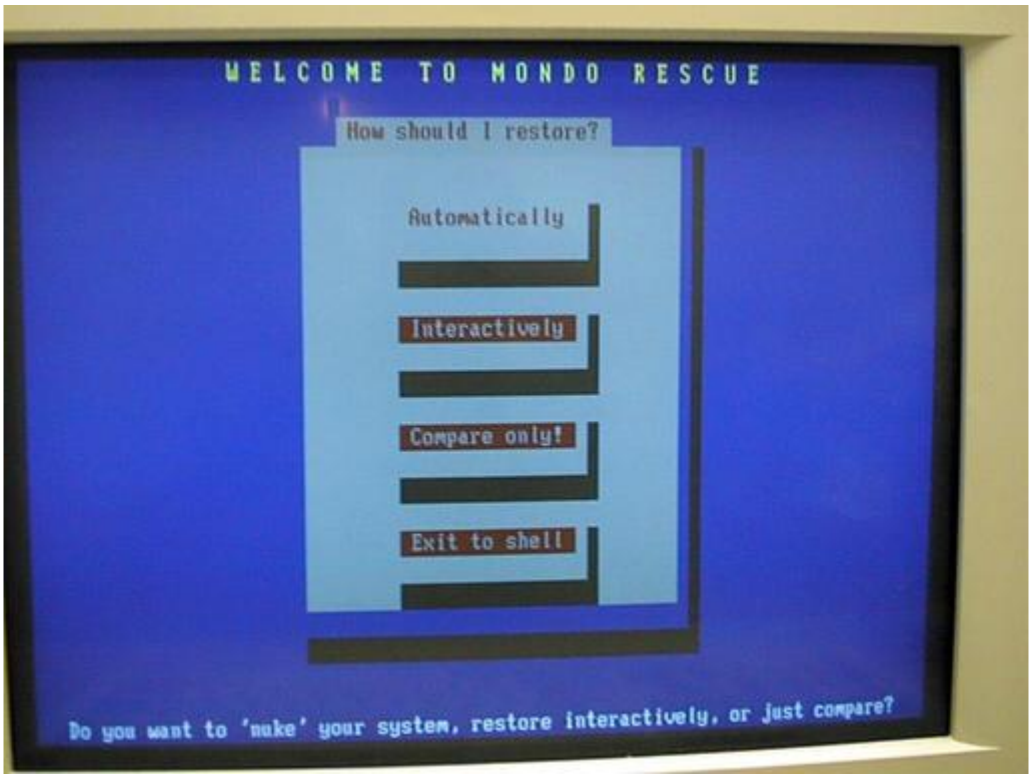
Restoration

You're probably saying, “Great, I have a bunch of CDs; now what?” Because the CD you made is bootable, you can pop it right in your CD-ROM drive and begin your mondoarchive restore learning experience. Four different restore options are available: barebones (nuke), interactive, expert and advanced.

If you're using mondoarchive solely as an emergency restore utility, the only option you really need to worry about is nuke. After the CD boots, type `nuke` at the prompt and watch mondoarchive do its magic. After it is done restoring, your system comes back to the way it was before you ran mondoarchive. This can be handy in the event of a catastrophic server failure when you need to get back up quickly. It also can be helpful when you're building preproduction or test bed systems that you would like to roll back to a certain development point.

The interactive mode of mondoarchive also can be useful. This mode enables you to edit your partition table before restoring your system, and it can come in handy when building clone machines with different hard disks. Using mondoarchive in interactive mode enables you to change the size of the partitions to which you want to restore data. This is possible using a partition table editor provided in mondoarchive. mondoarchive is even nice enough to resize your partitions when running in nuke mode, but here you don't have control over the sizes it uses. mondoarchive also resizes partitions in interactive mode, but you have the ability to change what it suggests.

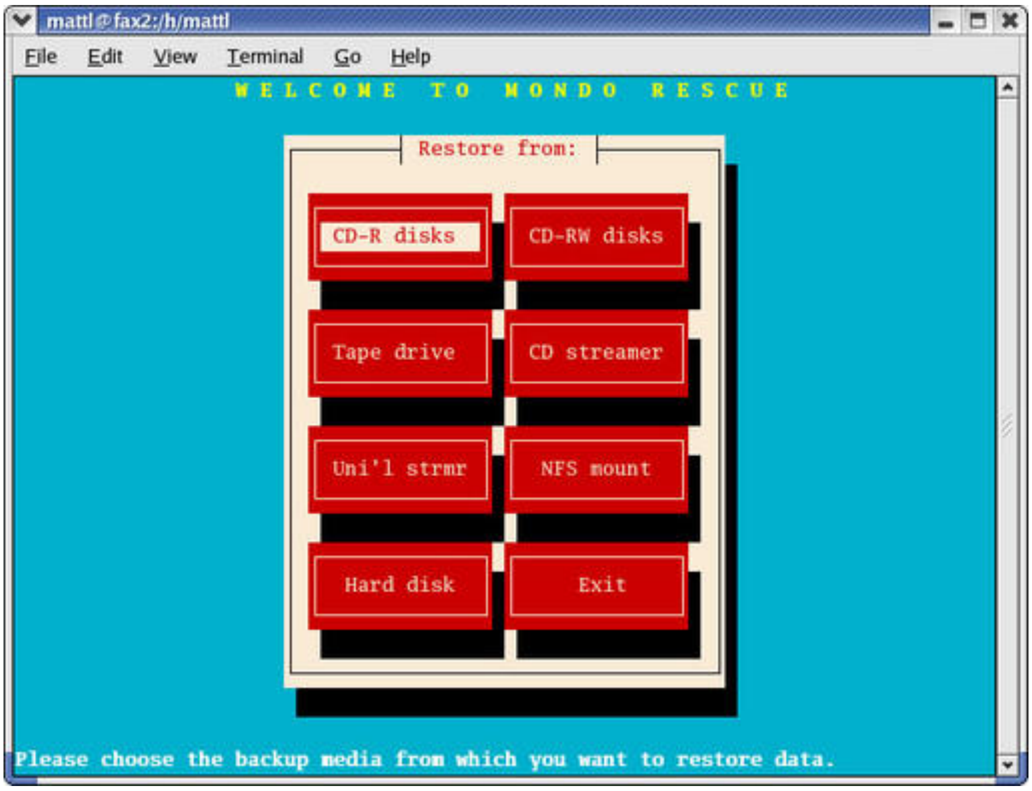
To use mondoarchive in interactive mode, boot to the CD and type `interactive`. A menu appears asking you how you want to restore (Figure 2).



Do you want to 'nuke' your system, restore interactively, or just compare?

Figure 2. Mondo's Interactive Mode

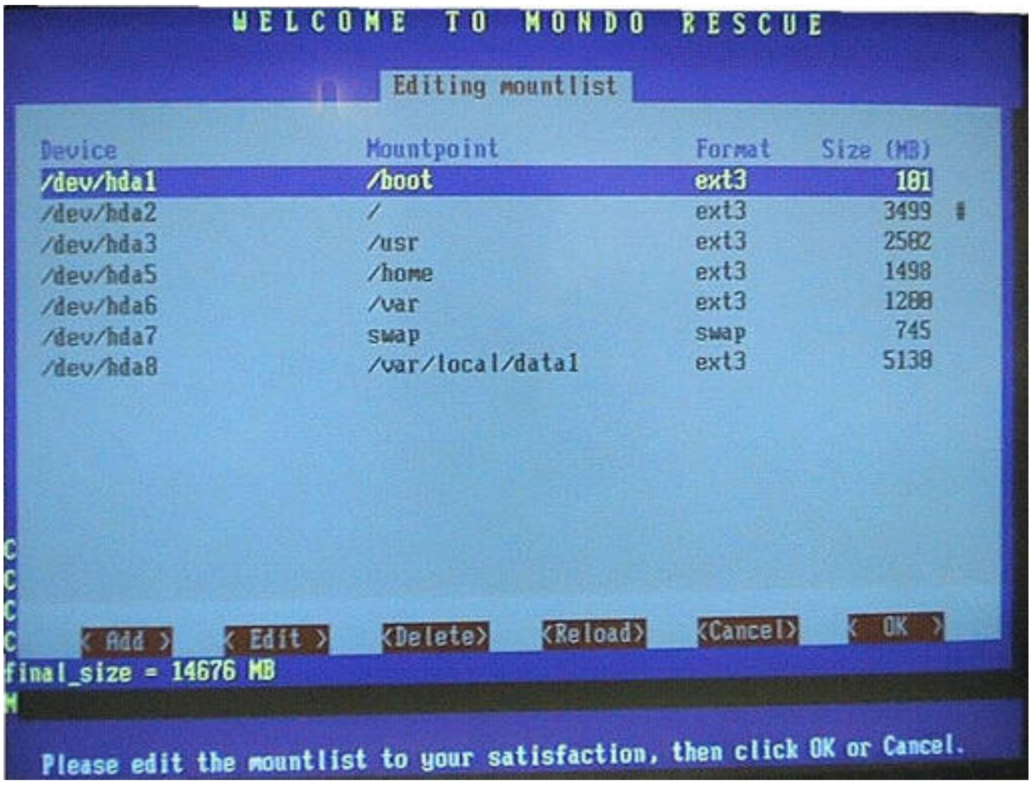
Your options are Automatically, Interactively and Compare. The option we want to select here is Interactively. After making the selection, you are asked where your data should come from (Figure 3).



Please choose the backup media from which you want to restore data.

Figure 3. Selecting the Data Source in Interactive Mode

Select the appropriate location and continue. Next, a partition manager screen appears (Figure 4) from which you have the ability to edit partition information. You also can change the device name, mountpoint, filesystem format and partition size. After making your changes, click OK to continue.



```
WELCOME TO MONDO RESCUE

Editing mountlist

Device             Mountpoint           Format    Size (MB)
/dev/hda1          /boot                ext3          101
/dev/hda2          /                    ext3         3499
/dev/hda3          /usr                 ext3         2502
/dev/hda5          /home                ext3         1498
/dev/hda6          /var                 ext3         1288
/dev/hda7          swap                 swap          745
/dev/hda8          /var/local/data1     ext3         5138

final_size = 14676 MB

Please edit the mountlist to your satisfaction, then click OK or Cancel.
```

Figure 4. Editing Partition Information in Interactive Mode

From this point on, Mondo asks you a series of questions regarding the restore operation:

```
  - Are you sure you want to save your
    mountlist and continue?  YES
  - Do you want to erase and partition
    your hard drives?  YES
  - Do you want me to restore all your data?  YES
  - Initialize the bootloader?  YES
  - Did you change the mountlist?  YES
  - You will now edit fstab and lilo.conf/grub.conf,
    to make sure they match your new mountlist.  OK
```

Mondo then opens up /etc/fstab and lilo.conf/grub.conf for editing in vi. You should make changes accordingly so they match your new mountlist:

```
  - Edit them again?  NO
  - Label your ext2 and ext3 partitions
    if necessary?  YES
```

When Mondo is finished, it throws you back to a prompt for reboot. Press Ctrl-Alt-Delete to reboot.

Verify a Software Load

In addition to a total restore option, mondoarchive also provides another function. In the event that the data on the machine you cloned is static, you probably don't want that data to change. This is the case at Midwest Tool & Die with our Linux routers. Using mondoarchive, we have the ability to verify a software load against the backup and discover whether any changes have been made recently. This is easy. First, reboot the system with CD #1 in the CD-ROM drive. Then type `# compare`.

When mondoarchive is finished, it puts the compare results in /tmp/changed.txt. There will be differences between the CD and the live filesystem you checked. Log files and most everything in /var, for example, are going to change all the time, so don't worry about them. What you probably want to look for are changes to anything located in directories such as /boot or /etc. The configuration files in those directories are system-specific and should not change much after the system is up and running. If you find changes to those files and you don't remember making them, you most likely have a problem.

The only problem with doing the compare is the fact that you have to bring down the system to verify it. Although this may not be a problem in some environments, you may be in a situation where your servers must be up 24 hours a day. In those cases, you probably are better off doing the verification during a scheduled downtime. An alternative to using mondoarchive to verify systems is the popular application Tripwire. I would recommend using Tripwire for static systems that need to be up all the time.

Conclusion

Now that you know the potential mondoarchive has for your disaster recovery plans, we recommend exploring it in more detail. In the event that something catastrophic occurs, you can recover quickly and completely. Please visit the Mondo Rescue home page at www.microwerks.net/~hugo/index.html and see what a great job Hugo Rabson, the author and maintainer of mondoarchive, does in maintaining mondoarchive. He is making our admin life a lot easier, and he could do the same for you.

Craig Swanson (craig.swanson@slssolutions.net) designs networks and offers Linux consulting at SLS Solutions. He also develops Linux software at Midwest Tool & Die. Craig has used Linux since 1993.

Matt Lung (matt.lung@slssolutions.net) works as a network engineer at Midwest Tool & Die. He also provides Linux consulting at SLS Solutions.

Archive Index Issue Table of Contents

Advanced search

Using the Amd Automounter

Erez Zadok

Issue #114, October 2003

How to use the Amd automounter to provide uniform and easily controlled access to all your file servers from anywhere.

Administrators today manage large sites with many Network File System (NFS) file servers and clients. Users like to be able to log in to any host and access the same files from any remote server. The naïve way of providing this access to all users is to hand mount all file servers on each client. Not only is this unwieldy to manage, but any unreachable server can cause all the clients to hang. Furthermore, users have to know what name to use to access files from certain servers.

The solution to these problems is an automounter. An automounter is configured with the knowledge of all file servers, such that administrators need only maintain the automounter's configuration in one place. Moreover, an automounter provides access to remote file servers on demand when users first try to access a pathname that leads to that server; this ensures that only servers in use and reachable are mounted on clients, minimizing the chances for hangs. Finally, the automounter provides a uniform namespace for the resources. For example, the pathname /src/kernel can hide the fact that the actual location of those files is server1:/n/raid/l/ksrc/v2.4/21preX.

Most commercial vendors and Linux distributions include an automounter. However, such automounters often work on only a single platform, use incompatible configurations or provide a limited feature set. The Amd Automounter, also known as the Berkeley Automounter, was developed with portability and a rich feature set in mind. It runs identically on numerous UNIX systems, provides a superset of features found in other automounters and supports a large set of features for even the most demanding administrator. If you are an administrator of a site with heterogeneous UNIX systems and want to make your life easier, Amd is the way to go.

This article explains how Amd works and provides a set of examples to demonstrate Amd's capabilities. The article is aimed at site administrators and anyone with a perversion for filesystems. We assume basic familiarity with Linux, NFS and filesystems. It is impossible to cover the Amd Automounter in detail in only a few pages; it would take a whole book to do that. Nevertheless, our examples are designed to start with the simpler, more common uses of Amd and gradually increase in complexity.

How Amd Works

Amd is a user-level dæmon that attaches itself to a directory, called an automount point. Amd receives requests from the kernel whenever users try to access that automount directory. Amd then ensures the actual resource the user is requesting is available and responds to the kernel's request. Finally, the kernel returns the appropriate status code to the user, and the user magically sees the desired files in the pathname requested.

Let's follow an example. Suppose Amd is started and attaches itself to the directory /src. A user runs `ls -l /src/kernel`. The kernel knows that the Amd dæmon is the valid file server for the /src directory, so the kernel suspends the user's ls process and sends a message to Amd asking it to resolve the name kernel within the automount point /src. When Amd starts, it loads automounter maps for each automount point. These automounter maps can be read from plain files, NIS/NIS+, LDAP, N/DBM and more. The map that Amd loads for /src is a list of key-value pairs. The key represents the name that Amd should provide within the automounted directory, and the value contains the information that Amd requires to resolve access to the named key. For our example, the map might contain:

```
kernel  type:=nfs;rhost:=server1;\
          rfs:=/n/raid/l/ksrc/v2.4/21preX
```

In this example, the key is `kernel` and it is separated from its value by whitespace. The value consists of three variable assignments, delimited by a semicolon. We use a back slash as a multiline continuation character. Variables are assigned values using the := Pascal-style syntax. Here, the type variable says the map entry describes an NFS mount. The rhost variable gives the name of the remote NFS server. Finally, the rfs variable describes the pathname of the exported directory on that remote host.

Going back to the suspended ls process, when Amd receives a lookup request from the kernel for the name kernel, Amd consults its map and finds the entry with that name. Amd mounts the /n/raid/l/ksrc/v2.4/21preX directory from the remote host server1. It then returns enough information back to the kernel about the type and actual location of that NFS-mounted directory for the kernel to resume the ls process. The ls process resumes listing the contents of the directory /src/kernel, unaware of the flurry of activity that transpired between Amd and the kernel, let alone where the actual location of the files for /src/kernel is.

Amd also checks to see when was the last time an automounted entry was used and automatically unmounts unused entries. This ensures that a system mounts only entries actively in use. In case you're wondering if you could control the timeout period, yes, you can. In fact, you can control dozens of parameters. Now you're wondering if you have to spend days configuring Amd. No, you don't; most parameters have been tuned carefully with appropriate defaults.

Amd Startup Configuration File

Amd uses a configuration file, often stored in /etc/amd.conf. The syntax of this file is similar to the smb.conf configuration file. Consider:

```
[global]
log_file = /var/log/amd
debug_options = all,noreaddir

[/net]
map_type = file
map_name = /etc/amd.net
mount_type = nfs

[/home]
map_type = nis
map_name = amd.users
mount_type = autofs
```

This amd.conf file first specifies global options that are applicable to all automounted directories. All options are simple key=value pairs. The first global option (log_file) specifies the pathname to a file for Amd to log information such as errors and trace activity. The second global option (debug_options) asks to turn on all verbose debugging other than the debugging associated with directory-reading operations. Next, we define two automounted directories. Here, Amd attaches and manages the directory /net, the entries for which come from the file /etc/amd.net. Amd also manages a /home automounted directory whose entries are read from the site's NIS (YP) server.

The mount_type parameter requires some background explanation. By default, Amd appears to the kernel as a user-level NFSv2/UDP server. That is, when the kernel has to inform Amd that a user has asked to look up an entry (for example, /src/kernel), the kernel sends RPC messages to Amd, encoding the NFS_LOOKUP request in the same manner that the kernel would contact any other remote NFS server. The only differences here are that Amd is a user-level process not a kernel-based NFS server, and Amd runs on the local host, so the kernel sends its NFS RPCs to 127.0.0.1. As a user-level NFS server, Amd is portable and works the same on every UNIX host. However, user-level NFS servers incur extra context switches and data copies with the kernel, slowing performance. Worse, if the Amd process were to die unexpectedly—which never happens, as our code is 100% bug-free—it can hang every process on the host that accesses an automounted directory, sometimes requiring a system reboot to clear.

A decade ago, Sun Microsystems realized these automounter deficiencies and devised a special in-kernel automounter helper filesystem called Autofs. Autofs provides most of the critical functionality that an automounter needs in the kernel, where the work can be done more reliably and quickly. Autofs often works in conjunction with a user-level automounter whose job is reduced to map lookup and parsing. Amd is flexible enough, as you can see from the above amd.conf example, to work concurrently as both a user-level NFS server and an Autofs-compliant automounter. All you have to do is set the mount_type parameter to the right value. So why not use Autofs all the time? Autofs unfortunately is not available on all operating systems, and on those systems where it is available (Linux, Solaris and a handful of others), it uses incompatible implementations that behave differently. For those reasons, not all administrators like to use Autofs. Nevertheless, with Amd you have the choice of which one to use.

User Home Directories

In almost every large site, user home directories are distributed over multiple file servers. Users find it particularly annoying when their home directories first exist in, say, /u/staff/serv1/ezk, and then—when new file servers are installed or data is migrated—the directories are moved to, say, /u/newraid3/ezk. A much better approach is to provide a uniform naming convention for all home directories, such that /home/ezk always points to the most current location of the user's home directory. Administrators could migrate a user's home directory to a new, larger file server and simply change the definition of the ezk entry in the amd.users map. Here's an example of a small amd.users map that mounts three users' home directories from two different servers:

```
#comment: amd.home map
/defaults  type:=nfs
ezk  rhost:=serv1;rfs:=/staffdisk/ezk
joe  rhost:=raid3;rfs:=/newdisk/joe
dan  rhost:=raid3;rfs:=/newdisk/dan
```

This example starts with a special entry called /defaults that defines values common to all entries in the map; here, all mounts in this map are NFS mounts. The subsequent three lines specify the user's name, plus the remote host and partition to mount to resolve the user's home directory. Although the pathname for each user's home directory, such as /home/joe, can remain fixed for a long time, the actual remote host and remote filesystem for Joe's home directory can change often without inconveniencing Joe.

As with Perl, there are several ways in Amd to achieve the same goal, and some ways are better than others. The above map is not the most optimal map for several reasons. So here are a couple of tips for optimizing Amd maps. First, consider what happens when you access /home/dan and are running on host raid3: Amd tries to perform an NFS mount of raid3 (as an NFS client) from raid3 as an NFS server. This is rather silly, going through the entire networking stack and the overhead of the NFS protocol, simply to get to a pathname local to the host. For that reason, Amd defines a different type of mount, a link type (using a symlink). Dan's map entry thus can be rewritten as:

```
dan  -rhost:=raid3;rfs:=/newdisk/dan \
      host!=${rhost};type:=nfs \
      host==${rhost};type:=link
```

This revised map entry introduces several new features of Amd maps. First, the back slashes are preceded by whitespace. Amd ignores whitespace after the back slash but not before; Dan's map entry essentially is broken into three distinct whitespace-delimited components called locations. The first location starts with the hyphen character and defines defaults for the map entry itself, overriding anything in /defaults. The second and third locations start with selectors. Amd map selectors are dynamic variables whose values could be compared at runtime by Amd. As you might expect from the mother of all automounters, Amd supports dozens of selectors. Amd evaluates Dan's map entry one location at a time until it finds one for which the selectors evaluate to true; Amd then mounts the given location. In order, Amd first compares whether the current running host's name does not equal the predefined value of rhost. On any host other than raid3, then, Amd performs an NFS-type mount. On raid3 itself, Amd uses a faster and simpler symlink-type mount.

The amd.home map contains a second inefficiency: it mounts /newdisk/joe and /newdisk/dan from the same NFS server, although they most likely are subdirectories of the same physically exported filesystem. This is slow and wastes kernel resources. A better way uses the same rfs but returns pathnames that are subdirectories of actual mountpoints (sublink is appended to returned pathnames automatically):

```
/defaults  type:=nfs;sublink:=${key}
joe  rhost:=raid3;rfs:=/newdisk
dan  rhost:=raid3;rfs:=/newdisk
```

/net Maps

A common use for automounter maps is to allow the mounting of any filesystem from any host, often called a net map. This map entry provides a comprehensive and uniform way of accessing all file servers:

```
   /defaults  fs:=/a/${rhost}/root/${rfs}
   *       rhost:=${key};type:=host;rfs:=/
```

This short example packs a lot of punch. First, we define the default fs variable's value to be a pathname that uniquely identifies the remote NFS server and filesystem being mounted. We can do this because the fs variable defines the pathname on the local host where Amd mounts remote hosts' filesystems. This pathname must be unique to avoid conflicts.

Second, the actual map entry's key, an asterisk, is a wild-card entry that matches anything and sets the key variable's value to the value of the name being looked up inside /net. This wild-card key value becomes the remote host's name (rhost). Next, we specify a special Amd mount entry of the type host, and we set the rfs variable to request mounting of all exported filesystems from that remote host (starting with /). The host mount type in Amd works by querying the remote host's rpc.mountd dæmon for the list of all exported filesystems. It then mounts each of them in turn. For example, suppose host foo exports two filesystems named /homes and /proj/X11. If you `chdir` to /net/foo, Amd mounts foo:/homes in /a/foo/root/home and mounts foo:/proj/X11 in /a/foo/root/proj/X11. An `ls` in /net/foo conveniently shows these two mounted directories.

One Map to Rule Them All

In large sites with many subgroups (sometimes with partial administrative control over the main group), there often are scenarios in which what you can mount depends on where you are. For efficiency, you may be limited on distributing binaries for different architectures to different subnets. With a single centralized set of Amd maps, it is possible to accommodate local needs:

```
   /defaults  type:=link
   lbin  in_network(eng);fs:=/local/${arch}/bin \
         in_network(10.0.1.0/24);fs:=/x/beta/bin
```

This map uses the in_network function selector. Function selectors evaluate to true or false based on the current system conditions and the parameters passed to these functions. This selector compares whether the current hostname is part of the eng network, often defined in /etc/networks. If so, Amd expands the value of the arch variable to the current running architecture. It also resolves the lbin entry to /local/i386/bin on IA-32 systems, to /local/sparc/

bin on SPARC systems and so on. The next location in this map shows that the in_network selector can match against network/netmask pairs. In fact, not only can this selector match using several forms, but it can match against any local interface up and running on your host. This capability offers the benefit of optimizing network routes on multihome hosts.

ISO-9660 Images

Many people keep ISO-9660 CD-ROM images handy, but accessing their content requires burning them onto a CD-ROM or mounting the images on Linux using a special loop driver. The cdfs mount type knows how to mount ISO-9660 CDs. But if you list a filename in the dev parameter and specify the loop mount option, Amd can mount those ISO image files directly. You then can browse them without copying the files out of each ISO image:

```
/defaults  type:=cdfs;opts:=loop
shrike1  dev:=/iso/rh9/shrike-disc1.iso
shrike2  dev:=/iso/rh9/shrike-disc2.iso
shrike3  dev:=/iso/rh9/shrike-disc3.iso
```

Do-It-Yourself Maps

Every self-respecting tool should have built-in extensibility mechanisms to accommodate the unexpected. For Amd to mount a given filesystem, it has to know how to mount it ahead of time (read: yours truly hacks the C sources). Using the program type mounts, you can define a custom method for mounting and unmounting any filesystem your native host knows about but which Amd doesn't:

```
r2  type:=program;dev:=/dev/sda1;\
      mount:="/sbin/dohans dohans ${dev} ${fs}";\
      unmount:="/sbin/undohans undohans ${fs}"
```

The above example passes the predefined dev parameter, as well as the automatically determined fs parameter to, say, a shell script named dohans that can perform whatever is needed to mount /dev/sda1 as a ReiserFS.

Conclusion

The Amd automounter is a powerful tool with many features to support numerous sites' needs. With proper care, site administrators can provide many useful features to users while reducing system administration efforts. Amd is part of the Am-utils distribution, which comes prebuilt with most Linux distributions. For more information, the latest sources, access to mailing lists and on-line documentation, visit www.am-utils.org. Happy automounting.

Erez Zadok (ezk@cs.stonybrook.edu) is on the Computer Science faculty at Stony Brook University, conducting filesystem and OS research. Erez is the author of *Linux NFS and Automounter Administration* (Sybex, 2001) and for the past ten years, the primary maintainer of Am-utils. Maintaining Am-utils is helped greatly by three co-maintainers: Ion Badulescu, Nick Williams and Rainer Orth.

Archive Index Issue Table of Contents

Advanced search

Securing Your Network against Kazaa

Chris Lowth

Issue #114, October 2003

The Kazaa peer-to-peer system is sneaky in getting around firewalls, but not sneaky enough.

Kazaa is the most popular file-sharing application in use today. Applications like it are known as peer-to-peer, or P2P, and allow users to search for and download files from each other. Kazaa apparently is used most often for sharing audio files in violation of copyright law.

Kazaa's proprietary network protocol, known as FastTrack, has been licensed to the developers of a number of similar products, including iMesh and Grokster. A stripped-down version of Kazaa called KazaaLite also is available. Plenty of other P2P applications exist, but the FastTrack family is by far the most popular, as well as the most difficult to block with packet-filtering firewalls, such as Linux's iptables.

Many network managers would like to block P2P traffic at their firewalls because of its high bandwidth usage, the security implications of uncontrolled file exchanges and potential legal action by copyright holders. This is not as easy as it might sound. A search on the Internet for information on blocking FastTrack traffic using iptables yields answers like “block port 1214”, “write a policy and discipline miscreants” or “it can't be done”. Blocking port 1214 used to work with early versions of FastTrack but doesn't with recent ones. Something more sophisticated is required. Although some “proxying” firewalls are able to block FastTrack traffic, iptables-based firewalls have issues that need resolving.

This article introduces a new open-source project called P2Pwall that develops software for preventing P2P clients on your network from making contact with peers on the outside. Its ftwall component blocks FastTrack traffic. More components will be written in due course to control other P2P protocols, and we invite you to become involved as a developer. The software has been tested with the following FastTrack clients: Kazaa 2.1.1, Kazaa 2.5, KazaaLite 2.0.2, iMesh 4.1 (build 132) and Grokster 1.7.

Firewalls Struggle with FastTrack

Modern Linux distributions include Netfilter and the iptables utilities. These components work together to allow Linux systems to be used as simple but effective firewalls; however, the FastTrack network protocol presents them with some interesting challenges:

- It doesn't use fixed port numbers.
- It is not limited to conversing with a small number of peers. It holds a cache of 200 peer addresses and tries to connect to all of them when it starts. The list changes regularly and is different on every machine.
- The peer-finding logic is not dependent on a central directory.
- Key parts of the protocol employ strong encryption.

Firewalls traditionally use one of two philosophies. The first is strict and blocks all ports except specific ones as required. The second is permissive and asymmetric and allows almost unlimited outbound connections while blocking almost all inbound ones. With both of these approaches, the port-agile FastTrack seeks out and exploits legitimately open ports. It can even exploit port 80. The strict paradigm plus a port-80 proxy is required to block FastTrack, but this approach is too restrictive for networks that want to retain a permissive paradigm while blocking P2P traffic.

The P2Pwall Project's ftwall Program

The P2Pwall Project aims to address these issues by providing a number of tools and documents that enable the filtering of P2P traffic. The FastTrack filter ftwall is the first such tool and is available for download under the GPL from p2pwall.sourceforge.net. ftwall interacts with iptables using the QUEUE target. It analyses the packets being forwarded through the firewall and decides whether they should be forwarded or discarded, based on an understanding of the characteristics of the FastTrack protocol. It tries to prevent any FastTrack traffic from leaving, and hence entering, the network.

ftwall's role is to block outbound FastTrack connections only on the assumption that inbound connections are already blocked by iptables rules. Many firewalls already use blanket blocks on inbound connections with a limited number of server connections enabled. However, if a FastTrack client on the inside connects to a peer on the outside, the outsider can call back in to the insider over the established connection. So, if we can rely on the firewall to block inbound connections and on ftwall to block outbound ones, we have a solution; however, we need to have both bits in place.

Installing and configuring ftwall is a matter of downloading the sources, compiling them and writing a few iptables rules. A possible complication is that one optional enhancement to the logic requires the ip_string module to be present in the kernel. The module currently is considered experimental and therefore is not included in many Linux distributions. You probably will have to add it yourself if you want to use it. See the P2Pwall Web site for more information.

The iptables QUEUE Target

When an iptables rule specifies QUEUE as a target, any packets matched by the rule are put into a queue for collection by an application such as ftwall. The program can then drop the packet or pass it back to Netfilter for further checking and forwarding. A typical rule for invoking this mechanism looks like this:

```
   iptables -A FORWARD -p tcp -i eth0 -dport 123 \
        -syn -j QUEUE
```

With this rule in place, all SYN packets from the network connected to eth0 and destined for port 123 on a remote host are passed to the program first. The program reads the packets and returns its verdict using the libipq library and ip_queue module.

QUEUE is a standard part of the iptables software delivered with most popular distributions. To verify that it is available on your system, type `insmod ip_queue` and check that no error message is displayed. For more details, see the Netfilter FAQ at www.netfilter.org/documentation/FAQ/netfilter-faq-4.html.

How ftwall Works

In order to explain the workings of ftwall, the description needs to go hand in hand with a partial explanation of FastTrack's connection logic. FastTrack connects to peers using three distinct approaches: a flood of UDP packets, parallel TCP/IP connections and a more traditional TCP/IP connection pattern. The software switches between modes if it believes it is being blocked. ftwall endeavors to keep clients running in the first mode for as long as possible, because this is the easiest to identify and allows a list of the peer addresses to be built up.

When a client starts, it sends large numbers of UDP packets through the firewall that are identifiable by their length and content. Netfilter queues these for processing by ftwall (Figure 1). Then, ftwall takes internal notes of the source and destination addresses of the packets and spoofs a reply to the client, thus preventing it from concluding that UDP packets are being blocked by the firewall and keeping it running in the first mode for a little longer.

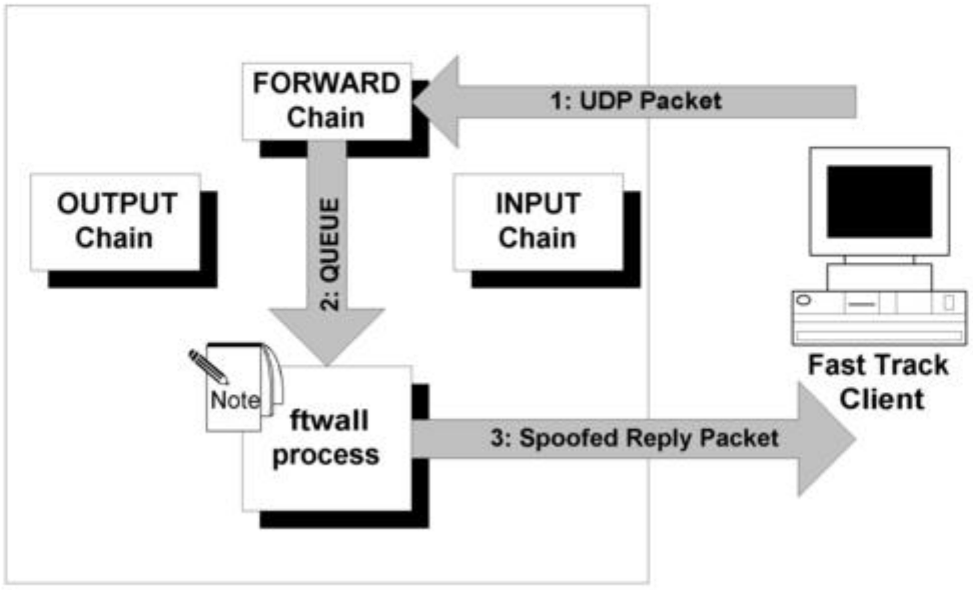


Figure 1. "openning shot" UDP packets

The iptables rule to set up this queuing, assuming eth0 is the home network interface, is:

```
iptables -A FORWARD -p udp -i eth0 -j QUEUE
```

When FastTrack receives the spoofed reply, it tries to use UDP to request some extra setup information and then attempts to make a TCP/IP connection to the same address. These UDP and TCP packets are passed to ftwall, which now knows that the destination addresses refer to FastTrack, and so it drops them (Figure 2). Other UDP non-FastTrack packets and TCP/IP SYN packets are returned to Netfilter for further checks and forwarded to their destination.

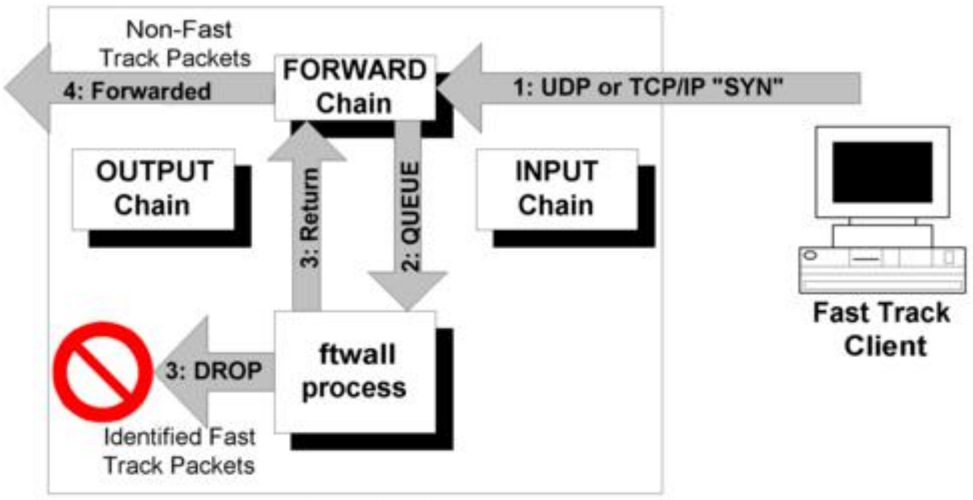


Figure 2. Other UDP and TCP/IP "SYN" Packets

The rule to queue SYNs to ftwall is:

```
   iptables -A FORWARD -p tcp -i eth0 --syn -j QUEUE
```

The client repeats this UDP and SYN sequence for a while—usually (but not always) until all the addresses it knows about have been attempted at least once. This means that all these addresses now also are known to ftwall as ones that should be filtered.

After a while, the client changes tack and switches to the parallel TCP/IP connection logic with strong data packet encryption. ftwall continues to block connections to addresses it noted during phase one. For any other addresses, the only clue that identifies them as FastTrack connections is the high number of SYN packets seen over a short period. If ftwall relied solely on the UDP packets to do the blocking, it would be defeated, particularly if the client hadn't tried all its known addresses in the first phase. The solution to this problem is a time lock.

In this new mode, the client mixes TCP/IP connection attempts to addresses that ftwall already knows about with others that haven't yet been revealed (if there are any). ftwall keeps a note of the time when the most recent known address was attempted and blocks all TCP/IP connections from the same source IP address for a configurable time after this. Each SYN packet sent to a known address resets the timer. Provided these connections are attempted frequently enough, ftwall continues to block them.

This logic has the side effect that all TCP/IP connections from a rogue workstation are blocked while FastTrack is running there, including accesses to Web and FTP sites. It can be argued that this is acceptable because the user of the workstation is breaking the organization's policy. Once the client application is closed, the timer ceases to be refreshed, and TCP/IP connections will be allowed again once it has expired. This takes two minutes with the default configuration.

After FastTrack has been working in this mode for a while, it appears to come to the conclusion that the parallel style of connection attempt is causing a problem, and it switches to its third mode. Now it slows down the rate of connection attempts and uses the more traditional approach of trying one address at a time, with a few seconds of timeout on each one. This new approach frustrates the logic we have built so far, and the client eventually breaks through. This can take over an hour to achieve, but clients that don't reveal all known addresses early on stand a reasonable chance of establishing a connection during this phase. And once a single connection is established, a completely new set of addresses is downloaded, and we are no better off than we would have been if no blocking was employed in the first place.

To defeat this third mode, ftwall needs more information to allow it to determine whether FastTrack is still in use. One way it can do this is with a little more spoofing. From time to time, ftwall sends the client a UDP packet that is a copy of the one that the client itself uses to open communications with a peer (Figure 3). If the FastTrack software is running on the workstation, it replies with a packet that can be recognized easily, thus causing the lock timer to be reset. The relatively small number and size of these probe packets means the impact on network usage is minimal.

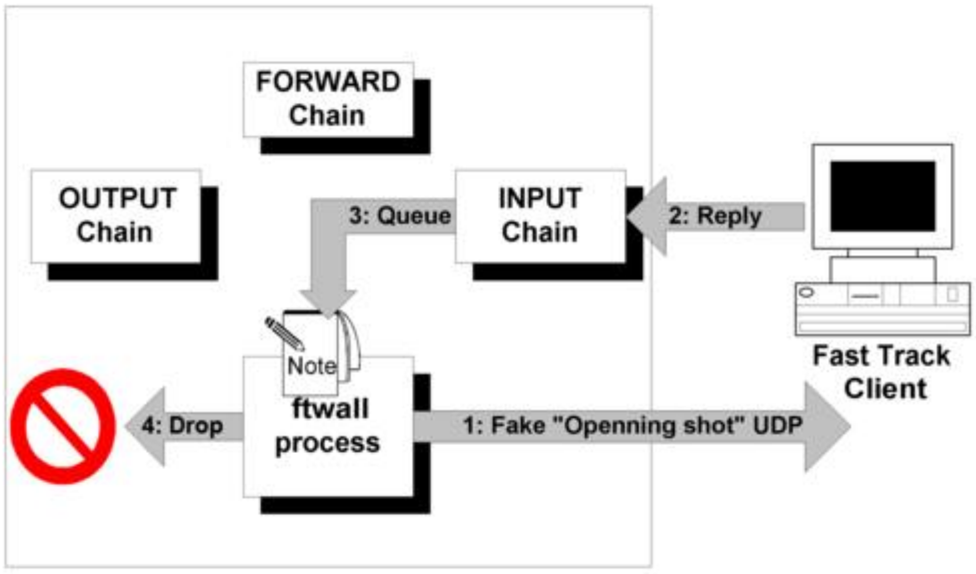


Figure 3. UDP Fast-track status probe

Because this packet is not for forwarding to a public address but destined for the firewall itself, an iptables rule in the INPUT chain is required to pass it to ftwall. The rule to use is:

```
iptables -A INPUT -p udp -i eth0 -j QUEUE
```

This keeps the client off-line permanently but is a little inefficient. If we choose the right time-lock timer, sending these UDP packets when it is half expired is all that is required to maintain the timer at a value that keeps the client blocked.

The final piece of the puzzle is a safety net that should not, in theory, be required. The logic described above depends on a set of recognizable UDP packets providing ftwall with the information it needs, but we need to consider what happens if these UDP packets don't arrive—if the user has disabled UDP transmission using the workstation's firewall software, for example. In this case, we have nothing that can be used to determine the addresses of peers being contacted.

However, we still have one option: inspect all TCP/IP data packets in an attempt to detect the actual transfer of files. FastTrack's use of encryption is limited to connection handshaking and searches. The shared files are transferred using clear-text HTTP, although this is not limited to port 80. The HTTP request headers include a number of fields that identify the FastTrack user, protocol and the address of a supernode, a node that provides index information. If these packets are queued for ftwall's inspection, it identifies those that look like the beginning of a FastTrack file download. From the information held in the HTTP headers, ftwall adds the target IP address and the supernode address to its list of blocked addresses and adds the client address to the list of those to which the time-lock logic is applied.

Installation Overview

The install process for ftwall is described in depth in the INSTALL file included with the software and on the project Web site, but in summary, the steps are as follows:

- Download sources from P2Pwall.sourceforge.net and unzip them.
- Install the libipq library, if it is not already installed. On some systems, including Red Hat 7.x and 8, this means obtaining the iptables sources and compiling them.
- Compile and install ftwall by running `make` and `make install`.
- Add an entry to the bootloader directory, /etc/rc3.d, to launch ftwall.
- Verify that the QUEUE mechanism is available, and add it if not. Most recent Linuxes already have this in place, but it can be added to those that don't by patching and rebuilding the kernel.
- Create the iptables rules in the INPUT and FORWARD chains.
- If you want to invoke the belt and braces option of inspecting the HTTP headers of the file downloads in case UDP is blocked on your network, add the string module to the kernel and iptables as well. This involves a kernel patch and rebuild.
- Reboot.

Conclusion

With ftwall running on the firewall, FastTrack traffic is blocked from reaching the Internet. Provided your firewall also blocks inbound connections, your network is Kazaa-proof. FastTrack clients in the network still can talk to each other, but file sharing with external peers is prevented.

This approach has the limitation of being focused solely on FastTrack; however, the P2Pwall Project aims to extend its reach to address other P2P protocols in the future. If you want to get involved with the project in any way, please e-mail me at chris@lowth.com.

ftwall works with the FastTrack clients available at the time of this writing. It is possible the FastTrack protocol will change in future, in which case ftwall may need to be modified to match.

Chris Lowth (chris@lowth.com) works for Intercai Mondiale (www.intercai.co.uk), a UK-based telecommunications, IT and business consultancy. He lives with his wife, three sons and golden Labrador in London, England. He plays the guitar, designs Linux-based security software, enjoys a good thunderstorm more than sun bathing and maintains body weight following a strict diet of French cheese and Indian cuisine.

Archive Index Issue Table of Contents

Advanced search

Building a Linux IPv6 DNS Server

David Gordon

Ibrahim Haddad

Issue #114, October 2003

A tutorial on building a DNS server node that provides IPv6 name resolution, with examples of some useful IPv6 applications.

IPv6 is the next-generation protocol designed by the Internet Engineering Task Force (IETF) to replace IPv4, the current version of the Internet Protocol. IPv4 has been remarkably resilient. However, its initial design did not take into consideration several issues of importance today, such as a large address space, mobility, security, autoconfiguration and quality of service. To address these concerns, IETF has developed a suite of protocols and standards known as IPv6, which incorporates many of the concepts and proposed methods for updating IPv4. As a result, IPv6 fixes a number of problems in IPv4 and adds many improvements and features that cater to the future mobile Internet.

IPv6 is expected to replace IPv4 gradually, with the two coexisting for a number of years in a transition period. Servers will be dual stack, supporting both IPv4 and IPv6.

In this article, we look closely at IPv6 name resolution and provide a technical tutorial to help readers set up their own IPv6 Linux DNS servers to allow IPv6 name resolution using the latest version of BIND 9.x.

General Network Overview

In this section, we present a sample network scheme (Figure 1) with different IPv6 servers.

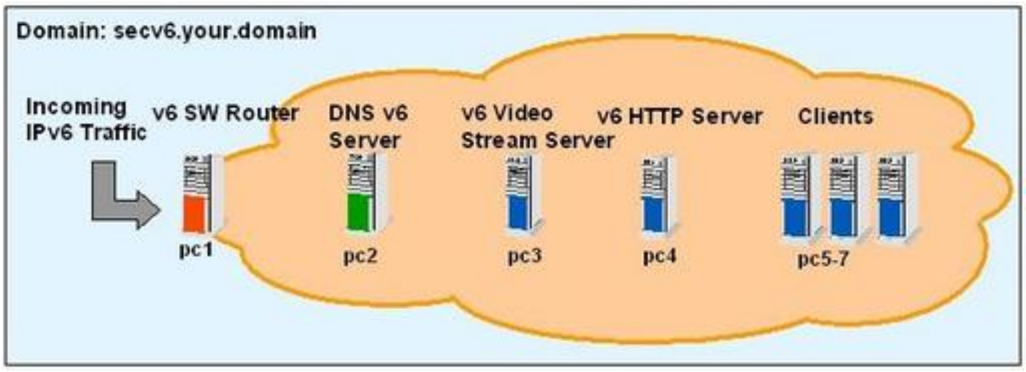


Figure 1. Sample Network Architecture

The following nodes are represented in this architecture:

- Routing server (pc1) acts as an IPv6 software router server and provides router advertisement for all IPv6 nodes.
- DNS IPv6 server (pc2) provides IPv6 name resolution.
- Two application servers, one provides video streaming (pc3) and the other is an Apache-based Web server (pc4).
- Client machines (pc5–7) used for testing purposes.

IPv6 Name Resolution

Domain names are a meaningful and easy-to-remember “handle” for Internet addresses. The domain name system (DNS) is the way that Internet domain names are located and translated into Internet protocol addresses. Because maintaining a central list of domain name/IP address correspondences is not practical, the lists of domain names and IP addresses are distributed throughout the Internet in a hierarchy of authority. Typically, a DNS server is within close geographic range of your access provider; this DNS server maps the domain names in DNS requests or forwards them to other servers on the Internet. For IPv6 DNS requests, both A6 and AAAA syntax are used to express IPv6 addresses.

AAAA resource record (called quad A record) is formatted as fixed-length data. With AAAA, we can define DNS records for IPv6 name resolution as follows, the same method as A records in IPv4:

```
$ORIGIN X.EXAMPLE.
N             AAAA 2345:00C1:CA11:0001:1234:5678:9ABC:DEF0
N             AAAA 2345:00D2:DA11:0001:1234:5678:9ABC:DEF0
N             AAAA 2345:000E:EB22:0001:1234:5678:9ABC:DEF0
```

An A6 resource record is formatted as variable-length data. With A6, it is possible to define an IPv6 address by using multiple DNS records. Here is an example taken from RFC 2874:

```
$ORIGIN X.EXAMPLE.
N             A6 64 ::1234:5678:9ABC:DEF0 SUBNET-1.IP6
SUBNET-1.IP6 A6 48 0:0:0:1::  IP6
IP6           A6 48 0::0       SUBSCRIBER-X.IP6.A.NET.
IP6           A6 48 0::0       SUBSCRIBER-X.IP6.B.NET.

SUBSCRIBER-X.IP6.A.NET. A6 40 0:0:0011:: A.NET.IP6.C.NET.
SUBSCRIBER-X.IP6.A.NET. A6 40 0:0:0011:: A.NET.IP6.D.NET.
SUBSCRIBER-X.IP6.B.NET. A6 40 0:0:0022:: B-NET.IP6.E.NET.
A.NET.IP6.C.NET. A6 28 0:0001:CA00:: C.NET.ALPHA-TLA.ORG.
A.NET.IP6.D.NET. A6 28 0:0002:DA00:: D.NET.ALPHA-TLA.ORG.
B-NET.IP6.E.NET. A6 32 0:0:EB00::    E.NET.ALPHA-TLA.ORG.
C.NET.ALPHA-TLA.ORG. A6 0 2345:00C0::
D.NET.ALPHA-TLA.ORG. A6 0 2345:00D0::
E.NET.ALPHA-TLA.ORG. A6 0 2345:000E::
```

If we translate the above code into AAAA records, it looks like:

```
$ORIGIN X.EXAMPLE.
N             AAAA 2345:00C1:CA11:0001:1234:5678:9ABC:DEF0
N             AAAA 2345:00D2:DA11:0001:1234:5678:9ABC:DEF0
N             AAAA 2345:000E:EB22:0001:1234:5678:9ABC:DEF0
```

Once IPv6 name resolution is configured, we can add domain name system (DNSSEC) to our DNS server. DNSSEC provides three distinct services: key distribution, data origin authentication and transaction and request authentication. The complete definition of DNSSEC is provided in RFC 2535.

Supporting IPv6 in the Kernel and in Network Binaries

An essential step prior to installing the IPv6-compliant BIND version is to enable IPv6 support in the kernel and for the networking binaries on the system supporting IPv6. We have covered this topic in a previous article, “Supporting IPv6 on a Linux Server Node”, in the August 2002 issue of *LJ* (/article/4763). After following the tutorial presented in that article, you should be ready to install the latest BIND version with IPv6 support.

BIND and IPv6 Support

The latest version of BIND is available from the Internet Software Consortium Web site (www.isc.org/products/BIND/BIND9.html). BIND version 9 is a major rewrite of nearly all aspects of the underlying BIND architecture. Many important features and enhancements were introduced in version 9; the most relevant to this article is the support for IPv6. BIND 9.x allows the DNS server to answer DNS queries on IPv6 sockets, provides support for IPv6 resource records (A6, DNAME and so on) and supports bitstring labels. In addition, BIND

9.x makes available an experimental IPv6 resolver library. Many other features are available, and you can read more about them from the BIND Web site.

Installing BIND 9.x

BIND 9.2.1 is the latest stable release available at the time of this writing. Our installation and configuration procedure follows this version. To install BIND, begin by downloading the latest BIND version into /usr/src, and then uncompress the package with:

```
% tar -xzf bind-9.2.1.tar.gz
% cd bind-9.2.1
```

Although IPv6 support is native to BIND, it must be specified explicitly when compiling. In addition, because we want to support DNSSEC, we need to compile BIND with crypto support. OpenSSL 0.9.5a or newer should be installed. Running the configuration script with the needed options looks like:

```
% ./configure -enable-ipv6 -with-openssl
```

Finally, compile and install the package as root with:

```
% make && make install
```

By default, the BIND 9 files are distributed in the filesystem. Configuration files are placed in /etc/named.conf; the binary “named” is in /usr/local/sbin and all other related configuration files go in /var/named.

Configuring IPv6 DNS and DNSSEC

DNS queries can be resolved in many different ways. For instance, a DNS server can use its cache to answer a query or contact other DNS servers on behalf of the client to resolve the name fully. When the DNS server receives a query, it first checks to see if it can answer it authoritatively, based on the resource record information contained in a locally configured zone on the server. If the queried name matches a corresponding resource record in the local zone information, the server answers authoritatively, using this information to resolve the queried name. For a complete DNS query process, there are four existing DNS zones:

1. Master: the server has the master copy of the zone data and provides authoritative answers for it.
2. Slave: a slave zone is a copy of a master zone. Each slave zone has a list of masters that it may query to receive updates to its copy of the zone. A slave, optionally, may keep a copy of the zone saved on disk to speed

startups. A single master server can have any number of slaves in order to distribute load.

3. Stub: a stub zone is much like a slave zone and behaves similarly, but it replicates only the NS records of a master zone rather than the whole zone. Stub zones keep track of which DNS servers are authoritative for the organization. They directly contact the root DNS server to determine which servers are authoritative for which domain.
4. Forward: a forward zone directs all queries in the zone to other servers. As such, it acts as a caching DNS server for a network. Or it can provide Internet DNS services to a network behind a firewall that limits outside DNS queries, but obviously the forwarding DNS server must have DNS access to the Internet. This situation is similar to the global forwarding facility but allows per-zone selection of forwarders.

To map this to our network (Figure 1), we need to create a master server for our own domain, secv6.your.domain. Listing 1 provides a sample /etc/named.conf configuration. (The secret key is truncated to fit on a line.)

Listing 1. /etc/named.conf

```
options {
directory "/var/named";

// a caching only nameserver config
zone "." IN {
type hint;
file "named.ca";
};

// this defines the loopback name lookup
zone "localhost" IN {
type master;
file "master/localhost.zone";
allow-update { none; };
};

// this defines the loopback reverse name lookup
zone "0.0.127.in-addr.arpa" IN {
type master;
file "master/localhost.rev";
allow-update { none; };
};

// This defines the secv6 domain name lookup
// Secure (signed) zone file is
// secv6.your.domain.signed
// Regular zone file is secv6.your.domain
zone "secv6.your.domain" IN {
type master;
file "master/secv6.your.domain.signed";
// file "master/secv6.your.domain";
};

// this defines the secv6 domain reverse
// name lookup (AAAA)
zone "secv6.int" IN {
type master;
file "master/secv6.int";
};

// this defines the secv6 domain reverse
// name lookup (A6)
zone "secv6.arpa" IN {
```

```
type master;
file "master/secv6.rev";
};

// secret key truncated to fit
key "key" {
algorithm hmac-md5;
secret "HxbmAnSO0quVxcxBDjmAmjrmhgDUVFcFNcfmHC";
};
```

The next step is to define the configuration files that describe our domain. Notice that until now we have not touched on the specifics of IPv6. As for DNSSEC, the file /var/named/master/secv6.your.domain.signed is the domain file signed by the zone key of the DNS server. This is important to DNSSEC, because clients are able to authenticate all subsequent DNS requests. The DNS server zone key is different from the key in the configuration file; the details on how to generate a zone key are discussed later in the article.

The next file to edit is /var/named/master/secv6.your.domain. Our example (Listing 2) uses both AAAA and A6 formats. The $INCLUDE directive at the end includes the public portion of the zone key. Keep the private portion of the key private. The private key has `private` appended at the end, whereas `key` postfixes the public key. If you have any concerns regarding DNSSEC keys and their permissions, consult the BIND manual. In Listing 2, we display a typical IPv6 DNS domain configuration for secv6.your.domain.

Listing 2. /var/named/master/secv6.your.domain

```
$TTL 86400
$ORIGIN secv6.your.domain.
@ IN SOA secv6.your.domain. hostmaster.your.domain. (
2002011442 ; Serial number (yyyymmdd-num)
3H ; Refresh
15M ; Retry
1W ; Expire
1D ) ; Minimum
IN MX 10 noah.your.domain.
IN NS ns.secv6.your.domain.
$ORIGIN secv6.your.domain.
ns 1D IN AAAA fec0::1:250:b7ff:fe14:35d0
1D IN A6 0 fec0::1:250:b7ff:fe14:35d0
secv6.your.domain. 1D IN AAAA fec0::1:250:b7ff:fe14:35d0 1D IN A6 0
fec0::1:250:b7ff:fe14:35d0
pc2 1D IN AAAA fec0::1:250:b7ff:fe14:35d0  1D IN A6 0
fec0::1:250:b7ff:fe14:35d0
pc3 1D IN A6 0 fec0::1:250:b9ff:fe00:131   1D IN AAAA
fec0::1:250:b9ff:fe00:131
pc6 1D IN A6 0 fec0::1:250:b7ff:fe14:3617  1D IN AAAA
fec0::1:250:b7ff:fe14:3617
pc4 1D IN A6 0 fec0::1:250:b7ff:fe14:35c4  1D IN AAAA
fec0::1:250:b7ff:fe14:35c4
pc5 1D IN A6 0 fec0::1:250:b7ff:fe14:361b  1D IN AAAA
fec0::1:250:b7ff:fe14:361b
pc7 1D IN A6 0 fec0::1:250:b7ff:fe14:365a  1D IN AAAA
fec0::1:250:b7ff:fe14:365a
pc1 1D IN A6 0 fec0::1:250:b9ff:fe00:12e   1D IN AAAA
fec0::1:250:b9ff:fe00:12e
pc1 1D IN A6 0 fec0:0:0:1::1 1D IN AAAA fec0:0:0:1::1
$INCLUDE "/var/named/master/Ksecv6.your.domain.+003+27034.key"
```

For configuration files in /var/named/master, Hostmaster actually is the e-mail address of the administrator, where the first dot replaces the at symbol (@) because of syntax restrictions. In addition, the first number for the IN SOA structure at the beginning of Listing 2 is the serial number conventionally expressed as YYYYMMDDNN, where NN is a number incremented each time the DNS zone is updated.

Now, we discuss how to generate a zone key. The working directory for this step is important because the keys are placed there. We suggest placing the keys in /var/named/master. The following command generates a 768-bit DSA key for the zone:

```
% dnssec-keygen -a DSA -b 768 -n ZONE \
secv6.your.domain
```

By default, all zone keys that have an available private key are used to generate signatures. The keys must be either in the working directory or included in the zone file. The following command signs the secv6.your.domain zone, assuming it is in a file called /var/named/master/secv6.your.domain:

```
% dnssec-signzone -o secv6.your.domain \
secv6.your.domain
```

One output file is produced: /var/named/master/secv6.your.domain.signed. This file should be referenced by /etc/named.conf as the input file for the zone.

The remaining configuration files are localhost.zone (Listing 3), localhost.rev (Listing 4), secv6.rev (Listing 5) and secv6.int (Listing 6). The difference between reverse lookup zone files secv6.rev and secv6.int is that one can be specified using A6 strings (that do not need to be reversed in secv6.rev) and the other with reverse AAAA format addresses in secv6.int. For instance, ping6 can refer only to secv6.int domain because it does not support A6 format.

Listing 3. /var/named/master/localhost.zone

```
// localhost.zone  Allows for local communications
// using the loopback interface
$TTL 86400
$ORIGIN localhost.
@ 1D IN SOA @ root (
42 ; serial (d. adams)
3H ; refresh
15M ; retry
1W ; expire
1D ) ; minimum
1D IN NS @
1D IN A 127.0.0.1
```

Listing 4. /var/named/master/localhost.rev

```
// localhost.rev  Defines reverse DNS lookup on
// loopback interface
$TTL 86400
$ORIGIN 0.0.127.in-addr.arpa.
@ IN SOA 0.0.127.in-addr.arpa. hostmaster.secv6.your.domain. (
42 ; Serial number (d. adams)
3H ; Refresh
15M ; Retry
1W ; Expire
1D ) ; Minimum
NS ns.secv6.your.domain.
MX 10 noah.ip6.your.domain.
PTR localhost.
```

Listing 5. /var/named/master/secv6.rev

```
// secv6.rev  Defines reverse lookup for secv6
// domain in A6 format
$TTL 86400
$ORIGIN secv6.arpa.
@ IN SOA secv6.arpa. hostmaster.secv6.your.domain. (
2002011442 ; Serial number (yyyymmdd-num)
3H ; Refresh
15M ; Retry
1W ; Expire
1D ) ; Minimum
NS ns.secv6.your.domain.
MX 10 noah.your.domain.
; fec0:0:0:1::/64
$ORIGIN \[xfec0000000000001/64].secv6.arpa.
\[x0250b7fffe1435d0/64] 1D IN PTR pc2.secv6.your.domain.
\[x0250b9fffe000131/64] 1D IN PTR pc3.secv6.your.domain.
\[x0250b7fffe143617/64] 1D IN PTR pc6.secv6.your.domain.
\[x0250b7fffe1435c4/64] 1D IN PTR pc4.secv6.your.domain.
\[x0250b7fffe14361b/64] 1D IN PTR pc5.secv6.your.domain.
\[x0250b7fffe14365a/64] 1D IN PTR pc7.secv6.your.domain.
\[x0250b9fffe00012e/64] 1D IN PTR pc1.secv6.your.domain.
```

Listing 6. /var/named/master/secv6.int

```
// secv6.int  Defines reverse lookup for secv6
// domain in AAA format
$TTL 86400
$ORIGIN secv6.int.
@ IN SOA secv6.int. hostmaster.secv6.your.domain. (
2002011442 ; Serial number (yyyymmdd-num)
3H ; Refresh
15M ; Retry
1W ; Expire
1D ) ; Minimum
NS ns.secv6.your.domain.
MX 10 noah.your.domain.
; fec0:0:0:1::/64
$ORIGIN 1.0.0.0.0.0.0.0.0.0.0.0.0.c.e.f.secv6.int.
0.d.5.3.4.1.e.f.f.f.7.b.0.5.2.0 IN PTR pc2.secv6.your.domain.
e.2.1.0.0.0.e.f.f.f.9.b.0.5.2.0 IN PTR pc1.secv6.your.domain.
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0 IN PTR pc1.secv6.your.domain.
1.3.1.0.0.0.e.f.f.f.9.b.0.5.2.0 IN PTR pc3.secv6.your.domain.
7.1.6.3.4.1.e.f.f.f.7.b.0.5.2.0 IN PTR pc6.secv6.your.domain.
4.c.5.3.4.1.e.f.f.f.7.b.0.5.2.0 IN PTR pc4.secv6.your.domain.
b.1.6.3.4.1.e.f.f.f.7.b.0.5.2.0 IN PTR pc5.secv6.your.domain.
```

Starting DNS Dæmon

Once the installation and configuration steps are complete, you are ready to start the DNS dæmon on pc2. Named uses /etc/named.conf by default, although you can specify a different configuration file with the -c option if you want. Depending on where you installed the dæmon, enter:

```
pc2% /usr/local/sbin/named
```

One additional configuration step is needed on the machines within the IPv6 network: update /etc/resolv.conf (Listing 7) to contain the DNS server's IP address. It is important that the IP address is included and not the hostname of the DNS server, because this file is where the system looks to find the address of the DNS. In other words, if you specified the hostname of the DNS server here, how would the system know what IP address corresponds to the DNS' hostname?

Listing 7. /etc/resolv.conf on Client Machines

```
# To enable secv6 domain, start named on pc2
# and use this file as /etc/resolv.conf
search secv6.your.domain
nameserver fec0::1:250:b7ff:fe14:35d0
```

Testing the Setup

We use two simple methods of testing the setup. The first verifies that A6 addresses are enabled in the DNS server, and the second verifies that AAAA addresses are supported by the DNS server. The tests were performed on pc2. We present only the meaningful output here; otherwise the listing would be too long. For the first example, we use the DNS lookup utility **dig** to perform a lookup on secv6 domain in A6 format (Listing 8). We then perform a lookup in AAAA format (Listing 9). In both cases, we are not specifying an address to look up, thus our use of 0.0.0.0.

Listing 8. A6 DNS Query

```
pc2% dig 0.0.0.0 secv6.your.domain a6
; <<>> DiG 9.1.0 <<>> 0.0.0.0 secv6.your.domain A6
[...]
;secv6.your.domain. IN A6
;; ANSWER SECTION:
secv6.your.domain. 86400 IN A6 0 fec0::1:250:b7ff:fe14:35d0
;; AUTHORITY SECTION:
secv6.your.domain. 86400 IN NS ns.secv6.your.domain.
;; ADDITIONAL SECTION:
ns.secv6.your.domain. 86400 IN A6 0 fec0::1:250:b7ff:fe14:35d0
ns.secv6.your.domain. 86400 IN AAAA fec0::1:250:b7ff:fe14:35d0
```

Listing 9. AAAA DNS Query

```
pc2% dig 0.0.0.0 secv6.your.domain aaaa
; <<>> DiG 9.1.0 <<>> 0.0.0.0 secv6.your.domain AAAA
[...]
;secv6.your.domain. IN AAAA
;; ANSWER SECTION:
secv6.your.domain. 86400 IN AAAA fec0::1:250:b7ff:fe14:35d0
;; AUTHORITY SECTION:
secv6.your.domain. 86400 IN NS ns.secv6.your.domain.
;; ADDITIONAL SECTION:
ns.secv6.your.domain. 86400 IN A6 0 fec0::1:250:b7ff:fe14:35d0
ns.secv6.your.domain. 86400 IN AAAA fec0::1:250:b7ff:fe14:35d0
```

For our second test, we include samples of an SSH session connection, first using an IPv6 address and then using an IPv6 hostname.

Sample Server Applications Using IPv6

In our IPv6 network, we presented two application servers: Apache as a Web server and VideoLan for video streaming. To test IPv6 name resolution when streaming a video, a user on client node pc5 accesses the video-streaming server on pc3. The video server is on pc3 (fec0::1:250:b7ff:fe14:5768), and the video client is on pc5 (fec0::1:250:b7ff:fe50:7c). Sniffing the network communications on pc5 with **tcpdump**, we captured packets from the video stream. Here is a portion of the trace:

```
% tcpdump ip6    # only trace IPv6 traffic, must be run as root or setuid root
[snip...]
02:09:26.716040 fec0::1:250:b7ff:fe14:5768.32769 > fec0::1:250:b7ff:fe50:7c.1234: udp 1316
02:09:26.735805 fec0::1:250:b7ff:fe14:5768.32769 > fec0::1:250:b7ff:fe50:7c.1234: udp 1316
02:09:26.735971 fec0::1:250:b7ff:fe14:5768.32769 > fec0::1:250:b7ff:fe50:7c.1234: udp 1316
02:09:26.736082 fec0::1:250:b7ff:fe14:5768.32769 > fec0::1:250:b7ff:fe50:7c.1234: udp 1316
02:09:26.755810 fec0::1:250:b7ff:fe14:5768.32769 > fec0::1:250:b7ff:fe50:7c.1234: udp 1316
02:09:26.755935 fec0::1:250:b7ff:fe14:5768.32769 > fec0::1:250:b7ff:fe50:7c.1234: udp 1316
02:09:26.775787 fec0::1:250:b7ff:fe14:5768.32769 > fec0::1:250:b7ff:fe50:7c.1234: udp 1316
```

The video is displayed properly using X11 output on a Linux X server; Figure 2 shows a capture from the stream.

Figure 2. The Output Stream of IPv6 Video

Conclusion

IPv6 is becoming a reality. For the next few years, we will need to be able to support both IPv4 and IPv6 on our servers before the complete transition to

IPv6 occurs. We need different pieces of the puzzle to achieve a full migration to IPv6, and one essential piece is an IPv6-compliant BIND implementation.

Acknowledgements

Ericsson Research Corporate Unit for approving the publication of this article. Open Systems Lab for supporting our work with Linux and IPv6. Simon Jubinville, Open Systems Lab, for his reviews.

Resources

BIND: www.isc.org/products/BIND/BIND9.html

BIND Manual: www.crt.se/dnssec/bind9/Bv9ARM.html

Comparison of AAAA and A6: www.ietf.org/proceedings/02mar/I-D/draft-ietf-dnsext-aaaa-a6-01.txt

DNSSEC: www.ietf.org/rfc/rfc2535.txt

DNSSEC and IPv6 A6: ftp.rfc-editor.org/in-notes/rfc3226.txt

DNSSEC Signing Authority: ftp.rfc-editor.org/in-notes/rfc3008.txt

IPv6 HOWTO: www.bieringer.de/linux/IPv6/IPv6-HOWTO/IPv6-HOWTO.html

IPv6 Linux Implementations: /article/5468

IPv6 Support for DNS: www.ietf.org/rfc/rfc2874.txt

IP Version 6 Addressing Architecture: www.rfc-editor.org/rfc/rfc2373.txt

Linux Kernel: www.kernel.org

“Supporting IPv6 on a Linux Server Node” by Hbrahim Haddad and Marc Blanchet, *LJ*, August 2002: /article/4763

David Gordon (David.Gordon@Ericsson.ca) is a computer science intern at Ericsson Research—Open Systems Lab. He is completing his undergraduate studies in Computer Science at Sherbrooke University. His research interests include security, next-generation IP networks and wireless technologies.

Ibrahim Haddad (Ibrahim.Haddad@Ericsson.com) is a researcher at the Ericsson Corporate Unit of Research in Montréal, Canada, involved with the system architecture of third-generation wireless IP networks.

Getting Started with Vi

William Ward

Issue #114, October 2003

Even if you use a different editor for most of your work, it helps to know the basics of the ubiquitous vi.

Most people who are getting started with Linux or UNIX today already know how to use graphical WYSIWYG (what you see is what you get) editors, with heavy use of the mouse, icons and pull-down menus. When they first encounter the traditional UNIX/Linux editor, vi, it may seem awkward and not very powerful, but the opposite is true.

As with graphical editors, you can select a range of text and execute a command to make changes, move the cursor by indicating where you want it to be and insert text by typing at the keyboard. The difference is that vi doesn't use the mouse; you use the keyboard to specify changes or move the cursor. In order for the entire keyboard to be available for commands, vi has separate command and insert modes.

Modes can be confusing at first. With most editors, whatever you type always goes into the document, and editing is done using a mouse or perhaps keyboard shortcuts. With vi, command mode uses the whole keyboard to specify editing commands. Only when you issue an insert command does the keyboard add text to the document.

Most new vi users learn only a few basic commands: the arrow keys, **i** for insert, **x** to delete a character, **dd** to delete a line, **:wq** to save and exit and perhaps a few others. They quickly grow tired of leaning on the same keys over and over but are afraid of learning a huge list of commands.

Instead of merely providing a list to memorize, here we explore the structure behind them, which makes the commands easier to remember and allows you to become a vi power user quickly. There are several basic categories of commands, including inserting text, moving the cursor, block edit commands, colon (ex) commands, options and miscellaneous commands.

Inserting Text

The first command to learns in vi is **i**. Typing **i** puts the editor into insert mode, where everything you type is inserted into the document until you press the Esc key. But **i** is not the only way to insert text.

To add text after the cursor, such as at the end of the line, use **a**. Or jump to the end of the line and append text with **A**. Similarly, **I** inserts at the beginning of the line. With these and other insert commands, always press Esc to return to command mode.

Use **o** to open a new line after the current one and insert text there or **O** to open before the current line.

Replace the current character with **s**, or replace many characters using **R**, similar to overwrite mode in a graphical editor.

Nearly all vi commands allow a numeric prefix, which usually repeats the command a specified number of times. For example, type `3iHooray` then press Enter and Esc. It inserts three lines of "Hooray" into your buffer. The **s** command is different; here, the number indicates how many characters are to be replaced: **5s** replaces five characters with whatever you type.

A few control keys have special meaning in insert mode. You can use the arrow keys to move around, but don't forget you're still in insert mode. Also, **Ctrl-T** and **Ctrl-D**, at the start of a line, can be used to indent. **Ctrl-T** moves to the next level of indentation, and **Ctrl-D** moves back. Use these with the autoindent and shiftwidth options (described in the Options section below) for best results.

Moving the Cursor

Graphical editors move around mainly by clicking the mouse. Some vi clones offer mouse support, but vi and all its clones share movement commands that can move you around quickly. The simplest are the arrow keys or the equivalent letters: **h** (left), **j** (down), **k** (up) and **l** (right). If these are all you use, however, you are missing out on the full power of vi.

You can move forward and back on a line by holding down the right or left arrow key (or **l** or **h**), but there are easier ways. Go to the beginning of the line with **^** or the end with **$**. To jump one word at a time, use **w** or **W** to go forward or **b** or **B** to go back. You also can use **e** or **E** to jump to the end of the current word. For lowercase **w**, **b** and **e**, a word is a sequence of alphanumeric characters; the uppercase versions use spaces to separate words.

Another way to move around on the current line is to jump to a particular character. For example, to move the cursor to the next s on a line, type **fs**. Use **F** for the previous one: **Fs**. Or use **ts** to go right before the s and **Ts** to go the other way. Change s to search for any character. Type **,** (comma) to repeat any of these searches.

To move the cursor to the top, middle or bottom of the screen, use **H**, **M** or **L** (uppercase). For **H** or **L**, a numeric prefix indicates how many lines from the top or bottom of the screen you want to go; it has no effect on **M**.

To jump back and forth by sentences, type **(** or **)**; for paragraphs, use **{** and **}**. If you're editing source code with a lot of brackets, try the **%** command: put the cursor on any of the following characters: (,), [,], { or }, and press **%**; you'll be taken to the matching bracket.

Often the best way to move around is by searching with the **/** command. For example, type `/hello` and press Enter to jump to the next “hello” in the document. Instead of “hello”, enter any regular expression. The **n** command repeats the search; **N** goes the other way. Or use **?** to search backward in the file. Search is case-sensitive; I discuss the ignorecase option later in this article.

With most movement commands, a numeric prefix can be used to repeat. For example, **5w** moves you forward five words, **2n** jumps not to the next search match, but to the one after that, and **5fs** jumps to the fifth s to the right of the cursor.

Block Edit Commands

In graphical editors, to change a block of text, click and drag the mouse to highlight the text, then click an icon or menu option or type a keyboard shortcut. Although vi does not utilize the mouse, the process is surprisingly similar.

As we have just seen, vi has a rich set of movement commands. Any of these will work as an argument to a block editing command (for example, **dw** to delete a word). In vi, instead of first highlighting with a mouse and choosing an action, type the block edit command (**d**) first, and *then* a movement command (**w**) to specify the block of text. As a shortcut, you can type the block edit command twice to affect only the current line, for example, **dd** deletes the current line.

As we've seen, **d** is for delete. Whenever you delete text, that text is remembered, much like the cut command in a graphical editor. To paste the text (“put” in vi parlance), type **p** to put it after the cursor's position (or uppercase **P** for before).

That takes care of cut and paste, but what about copy? In vi, it's called yank and is done with the **y** commands. These are exactly like the **d** commands, except they don't delete anything: **yy** yanks the current line, **yw** yanks a word and so on.

To replace a chunk of text, you could delete it first and use **i** to insert a replacement, or use **c** (change) instead, and you are placed into insert mode to type the replacement text; press Esc when done. Use with a movement command (such as, **cw**) or change the whole line with **cc**.

Programmers will love the **>** and **<** commands. These shift a range of text to the right or left. To shift a block of code, put the cursor on the { or } character at the beginning or end of the block and type **>%** to shift it to the right. Any other movement command can be used in place of **%**, or use **>>** to shift one line. Use **<** to shift left. See also **Ctrl-T** and **Ctrl-D** (while in insert mode) and the shiftwidth option below.

Lastly, use **!** to filter text through an external program. Type the program to run, and the selected part of the file is passed as standard input; the output of the program is inserted into the document in its place. For example, **!!** filters the current line; **!}** filters a paragraph. One command that is particularly useful with this is the UNIX/Linux filter **fmt**, which reformats paragraphs. To run an external program without modifying the file, see the **:!** command in the “Colon (ex) Commands” section.

Any numeric prefix given is passed along to the movement portion of the command, whether it is given first or between the edit and movement command. For example, either **3dw** or **d3w** deletes three words; **3yy** or **y3y** yanks (copies) three lines.

Colon (ex) Commands

Vi inherits as class of commands from its predecessor ex, accessible through what is sometimes called colon mode. Common examples are **:wq** (write and quit) and **:s/*X*/*Y*/** (substitute *Y* for *X*). Type **:** followed by an **ex** command, and press Enter. After the command is done, vi returns to command mode.

There are several variations on the **:w** and **:q** commands. To save the file and continue editing, use **:w**. You can quit vi with **:q**, but only if your work is saved; to quit no matter what, use **:q!** (dangerous). Try **:w!** to override read-only mode if **:q** gives an error. Many people use **:wq!** to write and exit. In vim or nvi, this is like typing **:w!** and then **:q** if there was no error while saving. But some other versions of vi treat it like **:w** and **:q!**, which means if there was any error saving the file, the work would be lost. Because of this, it is a better habit to use **:wq**.

The substitution command is extremely powerful. In its simplest form, **:s/*X*/*Y*/** replaces one occurrence of *X* with *Y* on the current line. To replace all occurrences on a line, add **g** (global) to the end, for example, **:s/*foo*/*bar*/g**. Add **c** to the end to confirm each change, for example, **:s/*foo*/*bar*/gc**. To change a range of lines, put two line numbers separated by a comma in front, for example, **:*10,20*s/*foo*/*bar*/g** replaces all occurrences of “foo” with “bar” on lines 10 through 20. Use 1 for the first line of the file and **$** for the last line; to replace on all lines, use **:*1*,$s/*foo*/*bar*/g** (instead of **1,$**, you can use **%**, as in **:%s/*foo*/*bar*/ g**).

Another useful command is **:g/*X*/**. It allows you to run any **ex** command for all lines matching the pattern *X*. For example, to print all lines that contain “hello” on the screen, use **:g/hello/p** (**:p** is an **ex** command for printing lines to the screen). You even can combine it with the **:s///** command. To change “foo” to “bar” on all lines that contain “hello”, use **:g/hello/s/foo/bar/g**.

In both **:s/*X*/*Y*/** and **:g/*X*/**, the expression *X* is a regular expression, not only a static string. Set the nomagic option to disable this (see below). To learn about regular expressions, I recommend the book *Mastering Regular Expressions* by Jeoffrey Friedl (O'Reilly & Associates, 2nd ed., 2002).

Use **:!** to run an external program. **:!ls** prints the current directory listing. As a convenience, the percent character (%) in the command is replaced with the name of the file you are currently editing, so use **:!chmod +x %** to make the file executable. Typing **:!!** repeats the previous **:!** command. Unlike the command-mode **!** commands, this does not modify the text; the output of the program, if any, is simply displayed.

Options

In vi you can set options to change the editor's behavior using the ex **:set** command. For example, the search commands are normally case-sensitive; to change this use the **:set ignorecase** or **:set ic** option. Most options have a two-letter abbreviated version.

There are two types of options, Boolean (true/false) options and those that take a value. To turn off a Boolean option, add “no” to the front; for example, **:set noignorecase** or **:set noic** turns off the ignorecase option.

For options that take a value, use the equal sign followed by the value to set. For example, the default spacing for the shift commands (**>**, **<**, **^T** and **^D**) is eight spaces (one tab character), but most programmers prefer two to four spaces. Use **:set shiftwidth=4** (or **:set sw=4**) to make all indentation commands use a four-space indent.

Here are some other useful options:

- **wrapmargin=*N*** turns on word wrapping; specify the number of characters before the end of the line at which to wrap, for example **:set wm=8**.
- **autoindent** (Boolean) indents each line you type the same amount as the previous line, for example, **:set ai**.
- **magic** (Boolean, default true, no abbreviation) controls regular expression behavior in the search commands, for example, **:set nomagic**.

To set several options at once, combine them into one **set** command like this: **:set noic wm=8 sw=4 nomagic ai**.

To save options for future editing sessions, put them in your .exrc file. When you run vi, it runs any ex (colon-mode) commands found in that file. Omit the : character, simply enter the **set** commands.

Miscellaneous Commands

A graphical editor's scrollbar tells you where you are, and you can click on it to view other parts of the file. Because vi has no scrollbar, it uses equivalent keyboard commands.

Ctrl-G displays the file status, including the current line number. Go to a particular line of the file with the ***X*G** command. **G** by itself jumps to the end of the file, or use **:*X*** as an alternate for ***X*G**.

Ctrl-F scrolls the screen one page forward, and **Ctrl-B** goes back one page. To move in half-page increments, use **Ctrl-D** (down) or **Ctrl-U** (up).

Some other miscellaneous commands:

- The **x** command deletes the current character (uppercase **X** deletes to the left). A numeric prefix deletes several characters. Text deleted using this command is cut as with the **d** commands.
- **~** changes the current letter from uppercase to lowercase or vice versa.
- Undo a mistake with the **u** command. Vim has multiple levels of undo. Repeating the **u** command undoes each step in turn; the **Ctrl-R** keystroke

puts them back (redo). Other versions of vi have only one level of undo; typing **u** again simply undoes the undo, restoring the change.

- An uppercase **U** undoes all changes made so far on the current line; once you leave that line, however, it doesn't work anymore.
- Type **.** (period) to repeat the last edit command.
- To repeat the last **:s///** command (but modifying only the current line), type **&**.

Read More

The vim editor comes with a handy tutorial to let you practice many of these commands. Type `vimtutor` from the shell prompt or `:help tutor` from within vim.

History of Vi

The vi editor (the name is short for "visual") initially was written in 1976 by Bill Joy at the University of California, Berkeley. It has been included in the BSD (Berkeley Standard Distribution) versions of UNIX ever since those days, and other versions of UNIX, including Linux, have adopted it over the years.

The vi editor is a descendant of ex, which in turn was based on ed. These older editors were designed for use on a teletype and could display only one line of text at a time; vi was the first UNIX full-screen text editor.

The original vi's source code has not been available due to copyright (until just a few months ago), but many clones have been written. The one included with most Linux distributions is vim (vi improved). The current standard BSD editor is nvi (new vi). Others include elvis, vile and stevie. These vi editors include the basic vi functionality, plus their own bells and whistles.

Resources

The Vi Editor and Its Clones: www.math.fu-berlin.de/~guckes/vi/index.php3 Vi Lovers Home Page: www.thomer.com/vi/vi.html

William R. Ward (wrw@bayview.com) has been using vi since 1987. He is a teacher, writer and programmer living in Mountain View, California with his wife, Holly. Their company, Bay View Consulting Services (www.bayview.com), offers training courses in Perl, Linux/UNIX and other related topics.

Archive Index Issue Table of Contents

Advanced search

Antique Film Effects with The GIMP

Eric Jeschke

Issue #114, October 2003

Try these effects to give your photos a classic and timeless appearance.

One of the much-touted advantages of digital cameras over film cameras is, at least at low ISO levels, a silky smooth absence of grain in images. Grain in traditional film prints or scans is the result of several factors, but the main culprit is the design of film itself: light-sensitive particles laid down in emulsion layers on celluloid. The larger these particles are, the more sensitive the film is to light and the faster it is considered to be. This determines the film's ISO rating. Higher sensitivity comes at a price; the larger the particles, the more easily they can be seen as grain in developed prints. Some high-speed black-and-white films, such as Kodak T-Max 3200, are especially grainy and are preferred by photographers who like the visual effect the grain structure adds to the photograph.

With digital, you can get film grain's counterpart, sensor noise. I usually want to reduce noise in my digital images, but sometimes I want that gritty, grainy film look, as seen in art houses, street photography and old photos. In this article, I describe a nifty trick for simulating film grain in an otherwise grainless digital image.

Another classic photography staple that has made the transition to digital imaging is sepia toning. Sepia toning originally was developed to extend the archival life of early black-and-white silver-based prints. The process has the effect of converting the silver grains in the print to silver sulphide, which is more stable than silver and thus slows the inexorable process of chemical deterioration. In the late 19th and early 20th centuries, it also provided a more pleasing color than straight black and white due to the poor quality of photo papers at the time, which were likely to be some shade of brownish-white.

In the traditional sepia-toning process, the developed print is gently agitated in a bleach solution to convert some or all of the silver from its metallic state.

After rinsing, the print is soaked in sepia toner until all the bleached silver is toned to the desired degree. Finally, the print is washed again to remove excess toner and then dried. With the transition to digital imaging and the use of archival pigmented inks, sepia toning is done these days almost exclusively for aesthetic reasons. A sepia photograph has a timeless, classic feel to it. With a powerful bitmap image-editing program such as The GIMP and a decent photo-quality color inkjet printer, you can achieve excellent quality sepia toning without the hassle of working up to your arms in toxic, smelly chemicals.

In this tutorial, I also show you how to accomplish the vignette effect, another popular holdover from the portrait world. Vignette in this case does not refer to a “gradually faded oval cutout”, one popular definition of the term. Rather, I mean, lens vignetting, where a special lens or lens hood is used to achieve a gradual light falloff toward the corners of the image frame.

Simulating Film Grain

Most of the menus in The GIMP are accessed by clicking the right-most mouse button in an image window. In the description that follows, a right-click is abbreviated RC. If I describe a GIMP action to invoke, I will mention the series of menus or a keyboard shortcut in parentheses. For example, open the image (RC→File→Open), means right-click in the image window and choose File, and from that menu choose Open. If a keyboard shortcut makes more sense, I list the combination of keys to press; for example, copy the image (Ctrl-C) means press and hold the Ctrl key and press C.

To start with, I assume you have a photo with artistic potential loaded into The GIMP, as shown in Figure 1.

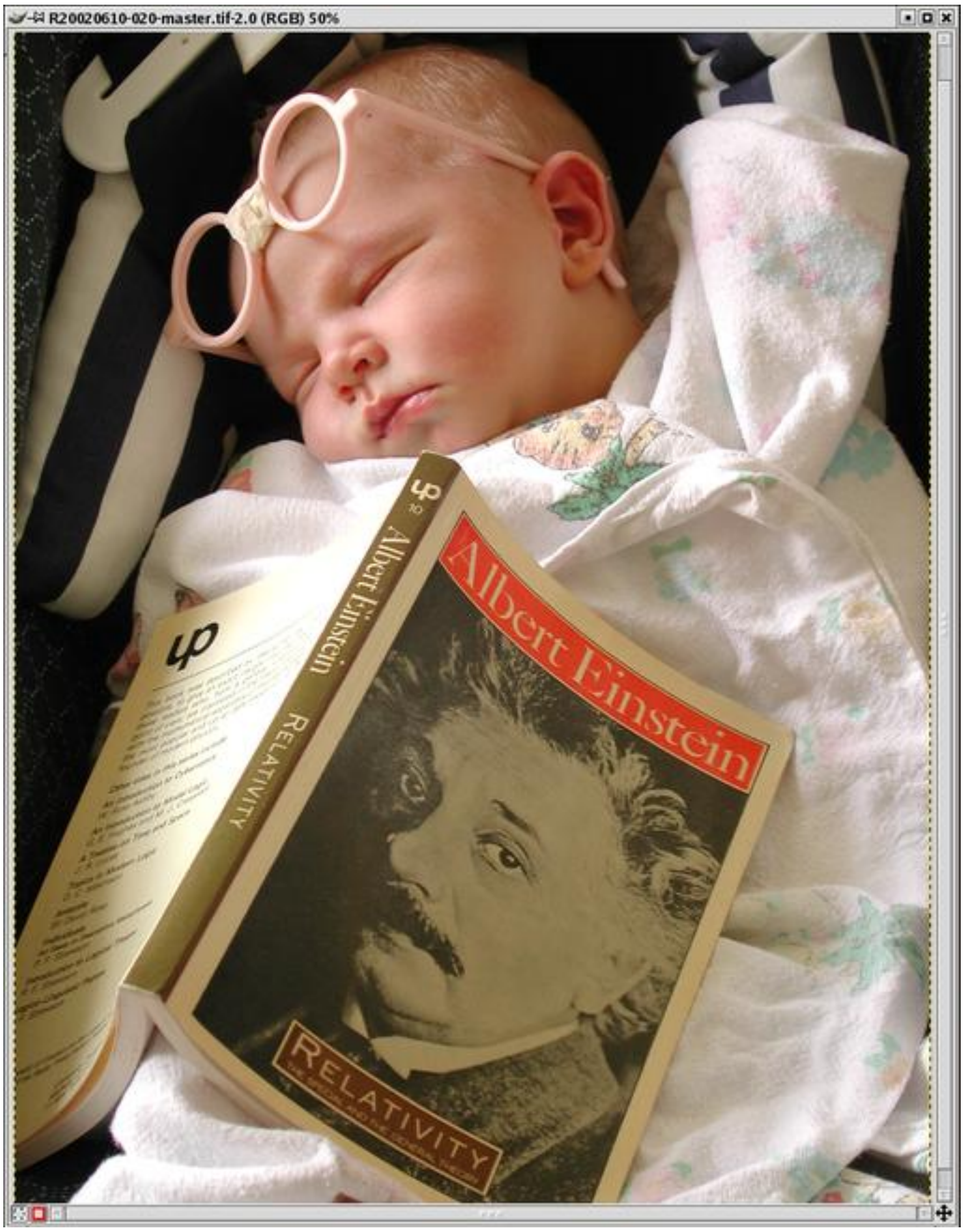


Figure 1. A Photo with Timeless and Classic Potential

If you are like me and want to use the effect in black and white, the first step is to convert it from color. An easy way to do the conversion is to perform a mode change to grayscale (RC→Image→Mode→Grayscale), which provides reasonably pleasing results for a large percentage of images. For some images, though, other approaches may yield better results. For the ultimate in flexibility, try a custom mix using the Channel Mixer plugin (RC→Filters→Colors→Channel Mixer), with the Monochrome option selected. A tutorial on converting to black and white is at mmmaybe.gimp.org/tutorials/Color2BW.

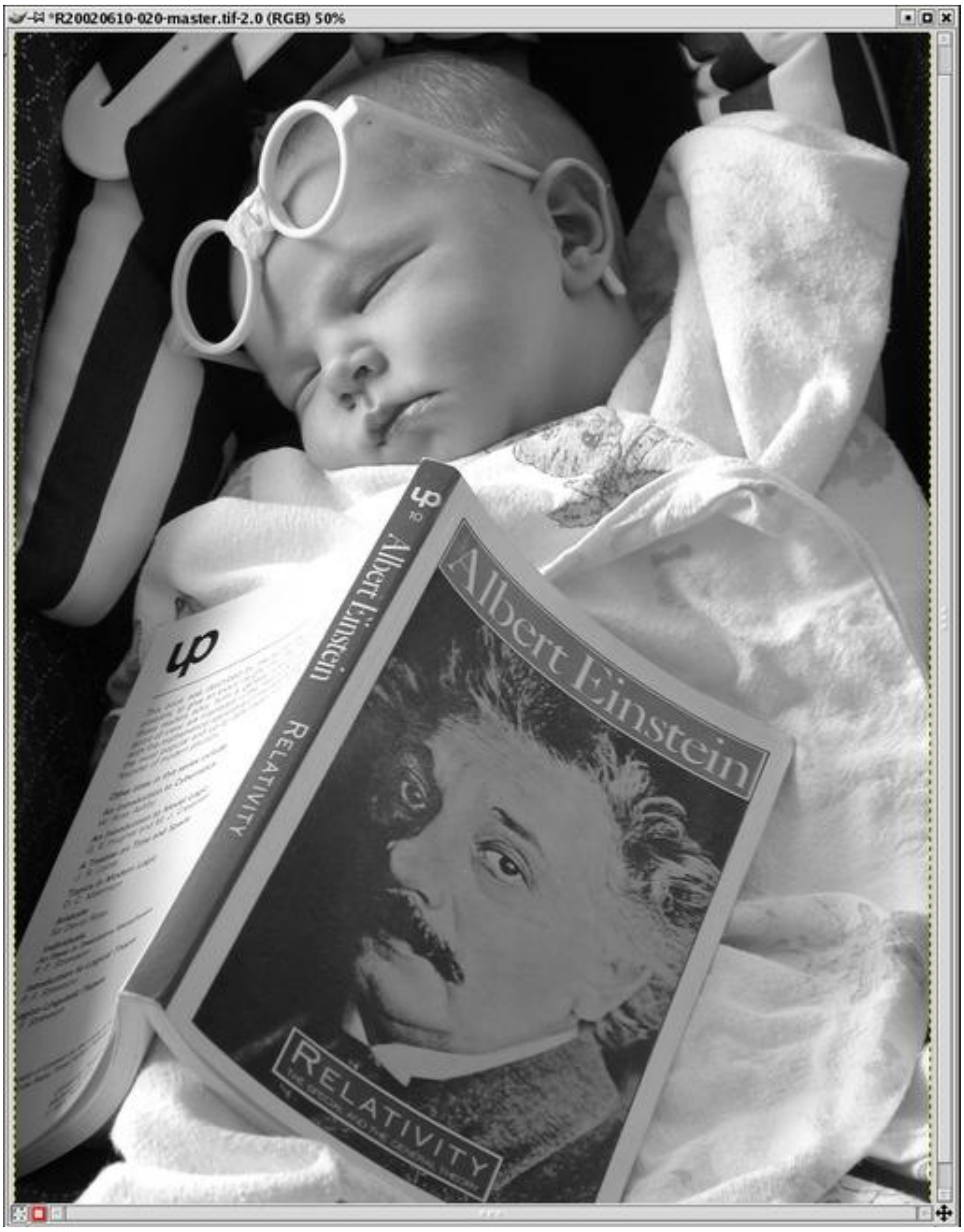
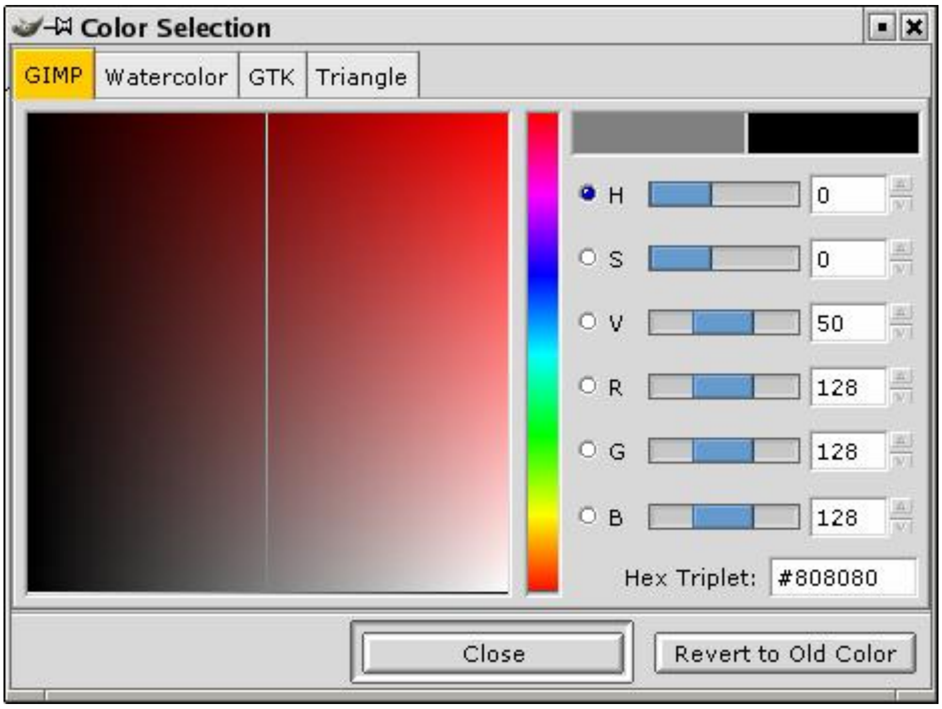


Figure 2. After Mode Change to Grayscale and Back to RGB

Once you have a decent-looking black-and-white image, you are ready to proceed. If you used a standard grayscale mode change, as I did here, you need to put the image back in RGB mode (RC→Image→Mode→RGB).

Click on the foreground color swatch in The GIMP toolbox to bring up the color selection dialog. Dial in the color red = 128, green = 128, blue = 128, as shown in Figure 3, and click OK. This selection should make the foreground swatch in the toolbox a neutral gray, as shown in Figure 4.



H 0
S 0
V 50
R 128
G 128
B 128
Hex Triplet: #808080

Figure 3. The Color Selection Dialog

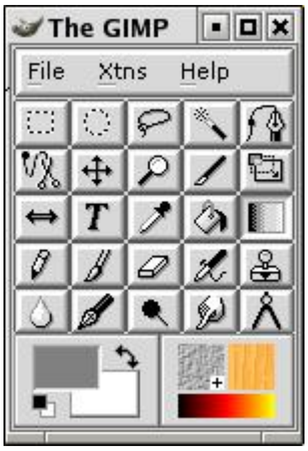


Figure 4. The GIMP Toolbox after Foreground Color Change

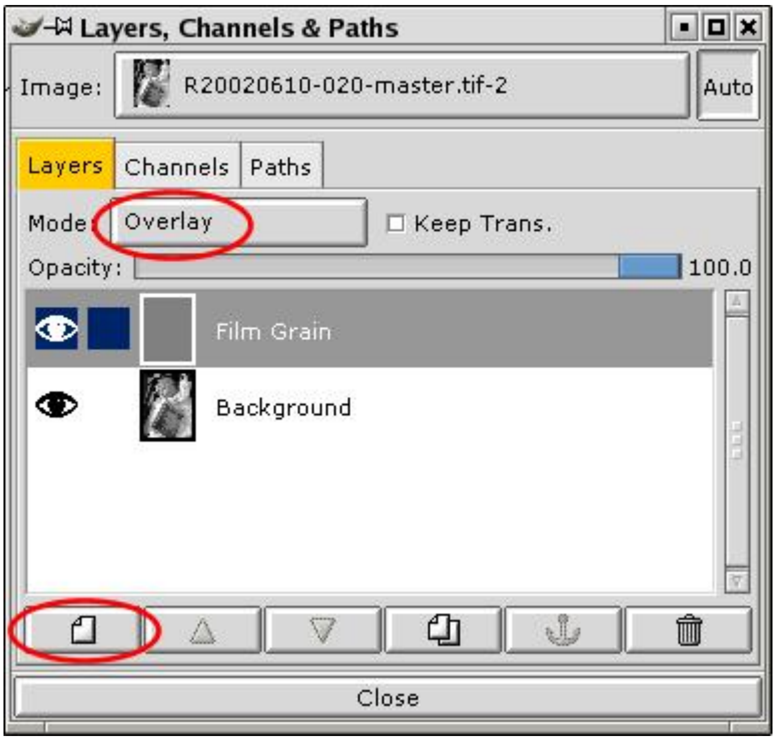


Figure 5. The Layers Dialog after Addition of the Film Grain Layer

Now bring up the Layers dialog (Ctrl-L) and click on the button for a new layer, circled at the bottom of Figure 5. Give it the name Film Grain and choose the option to fill it with the foreground color. Once you click OK, you should see nothing but a solid gray color in the image window as the newly created layer obscures the image in the layer below. Now, change the blending mode of the layer to Overlay (circled at the top of Figure 5), and you should see your image again. The blending mode is an attribute of a layer that describes how it combines with the layers below it to form a composite image. In Overlay mode, anything lighter than neutral gray lightens the image and anything darker darkens it. Our layer is completely neutral gray at this point, so there is no visible difference from the background image.

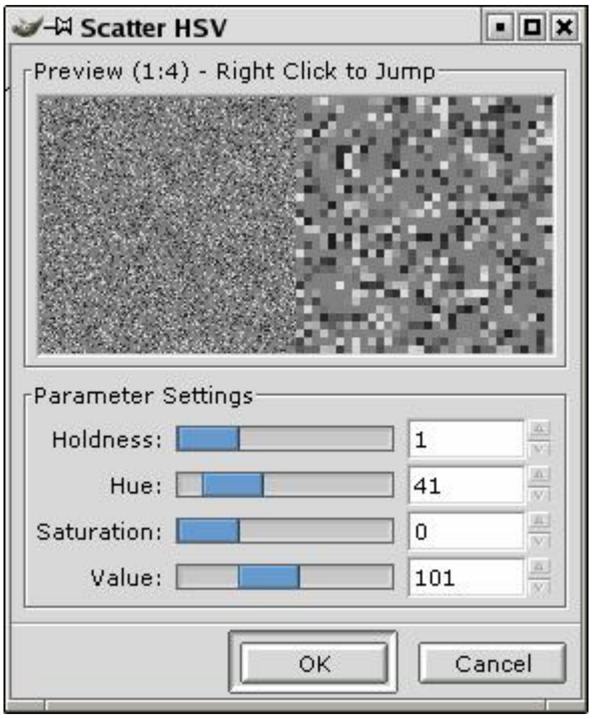
With the Film Grain layer selected in the Layers dialog, bring up the Scatter HSV filter by right-clicking in the image window and selecting Filters→Noise→Scatter HSV from the pop-up context menus. This filter adds a noise pattern to the neutral gray layer, which overlays to the image below it, creating a grain-like pattern.

Here's a not-too-technical interpretation of the filter parameters for the purpose of creating grain:

- Value: think of this as the master control for the granularity and intensity of the grain. Increasing this increases the contrast in the grain and the cluster size of some introduced noise, making the simulated grain seem larger and darker.

- Hue: if Saturation (see below) is set to 0, this has little effect other than to change the (random) pattern of the grain. Play with it until you see a pattern you like.
- Saturation: set this to 0 unless you want to add colored grain (a color image, for example). If colored grain is what you want, adjust this as well as the Hue control.
- Holdness: the fine-tune control for the granularity and intensity of the grain. Adjust this after you've made a change to the other controls. Higher amounts make the grain finer and less noticeable.

Play with the parameters until you see an interesting grain pattern in the preview window of the filter, as shown in Figure 6, and then click OK. When the filter finishes, you should see your image take on a grain pattern. If you don't like the look, Undo (Ctrl-Z) and reapply the filter with different settings (Shift-Alt-F). You probably don't want to go too light on the effect here, because the next steps will soften the grain and mute its effect.



Holdness: 1
Hue: 41
Saturation: 0
Value: 101

Figure 6. The Noise/Scatter HSV Filter Showing Typical Film Grain Parameters

Although you could stop at this step with decent results, there is room for improvement. If you examine a traditional grainy black-and-white photo, you can see that film grain tends to be most noticeable in the mid-tones and is much less so in the shadows and highlights. What we need is a way to control the blending so the highlights and shadows of the background layer get less of the grain effect. If you read my earlier tutorial on layers and layer masks (see “Fixing Photo Contrast with The GIMP”, *LJ*, April 2003), you might remember that we can control the blending of layers using a layer mask. The lighter a pixel is in the layer mask, the more opaque the corresponding pixel is in the upper layer;

the more opaque it is, the stronger its effect is when blending with a pixel in the layer below. The trick to achieving the desired effect is to make our layer mask be a half-inverted copy of the background image. Read on, and you'll see what I mean.

In the Layers dialog, right-click on the Film Grain layer and choose Add Layer Mask. In the Add Mask Options dialog, choose White (Full Opacity) and click OK. Now click on the Background layer. Return your mouse focus to the image window and do a select all and then copy (Ctrl-A followed by Ctrl-C is the quickest). Return to the Layers dialog, and click on the layer mask icon in the Film Grain layer; it's the little white square next to the layer thumbnail, circled in Figure 7. Then, move your mouse focus back to the image window and paste (Ctrl-V). In the Layers dialog, click the Anchor button to anchor the pasted image into the layer mask. After this step, your Layers dialog should resemble Figure 7.

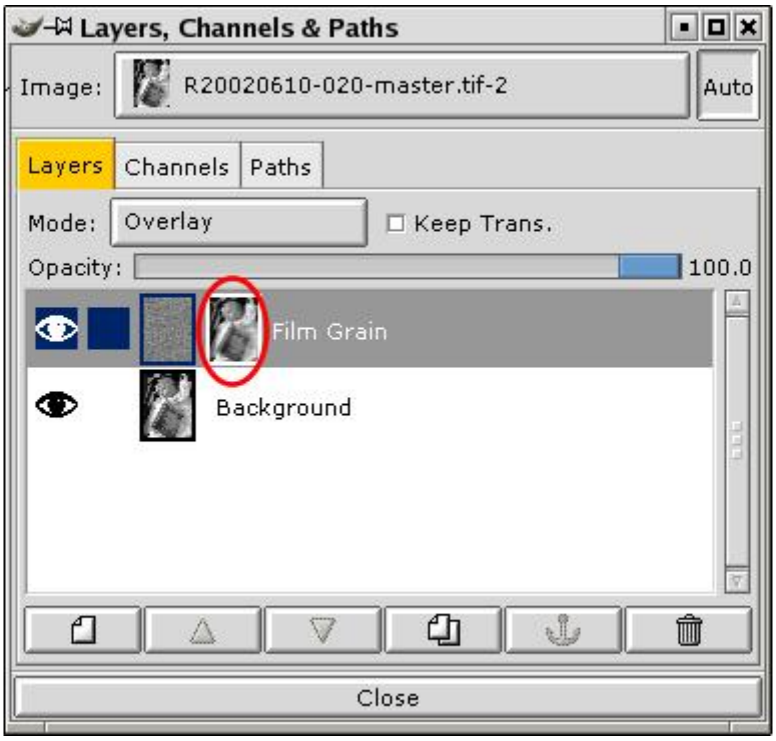
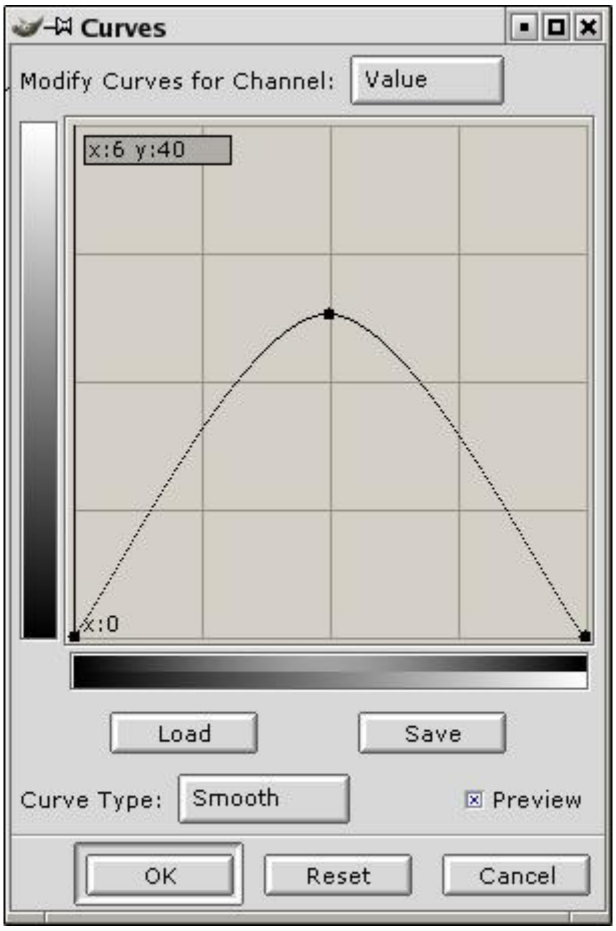


Figure 7. Adding a Layer Mask to the Film Grain Layer



Modify Curves for Channel: Value

Curve Type: Smooth

☒ Preview

Figure 8. The Curves Dialog, Half-Inverting the Layer Mask

With the layer mask still selected, if you just pasted into it, bring up the Curves dialog by right-clicking in the image window and selecting Image→Colors→Curves. Click to add a control point in the middle of the linear graph. Then grab the right (upper) endpoint and drag it down to the lower right bottom, as shown in Figure 8. What you are doing is half-inverting the layer mask, making all highlights into shadows, so the mid-tones are the brightest part of the image. You also may want to drag the midpoint straight up a bit as I have done here to boost the brightness of the midtones—keep an eye on the image window while you are doing this so you can see the effect of the boost on the grain. When you're all done click OK; you should see a subtle difference in the way the grain shows up in the shadows and highlights. Overall, the grain effect is subdued somewhat, which is why we didn't want to go too subtle in the previous step.

To see the effect of the layer mask more clearly, hold the Ctrl key and click the layer mask icon in the Layers dialog. A little red outline should display around the icon, and the image window changes to show the blend without the effect of the layer mask. Ctrl-click the icon again to toggle the effect of the mask back on.

As a last step, consider applying a bit of Gaussian Blur to smooth the grain a little. Click the grain icon in the Layers dialog to select it, then go back to the image window and RC→Filters→Blur→IIR Gaussian Blur. Be aware that if you use anything other than a very small blur radius you counteract the effect of adding the grain in the first place and end up with a noisy, but not noticeably grainy image. A value between 1 and 3 should be sufficient if you used a high Value setting in the Scatter HSV filter. If you like the grain the way it is, skip the blurring step. The result I obtained in Figure 9 used the HSV Scatter settings shown in Figure 6 and a Gaussian Blur with a radius of one pixel.

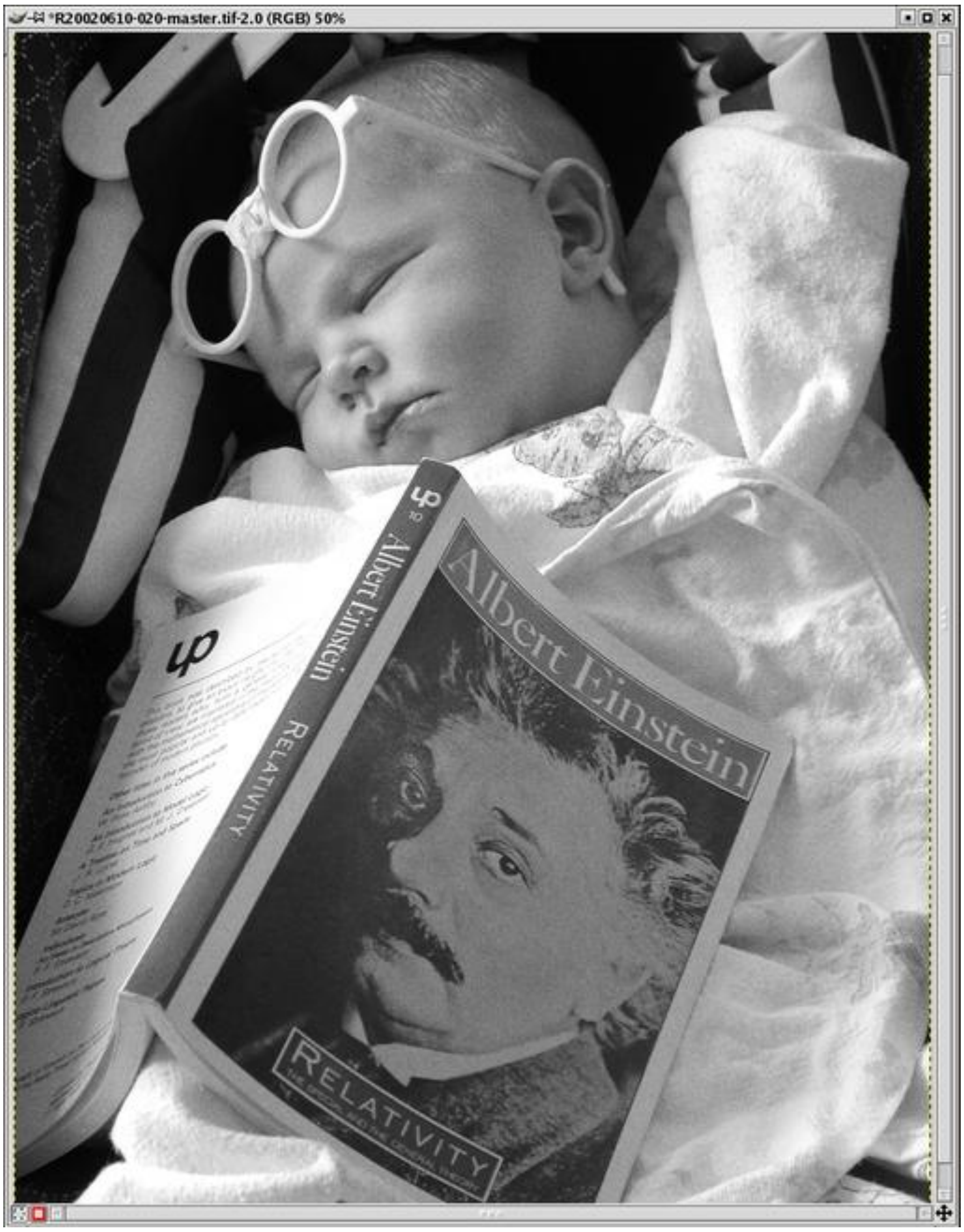


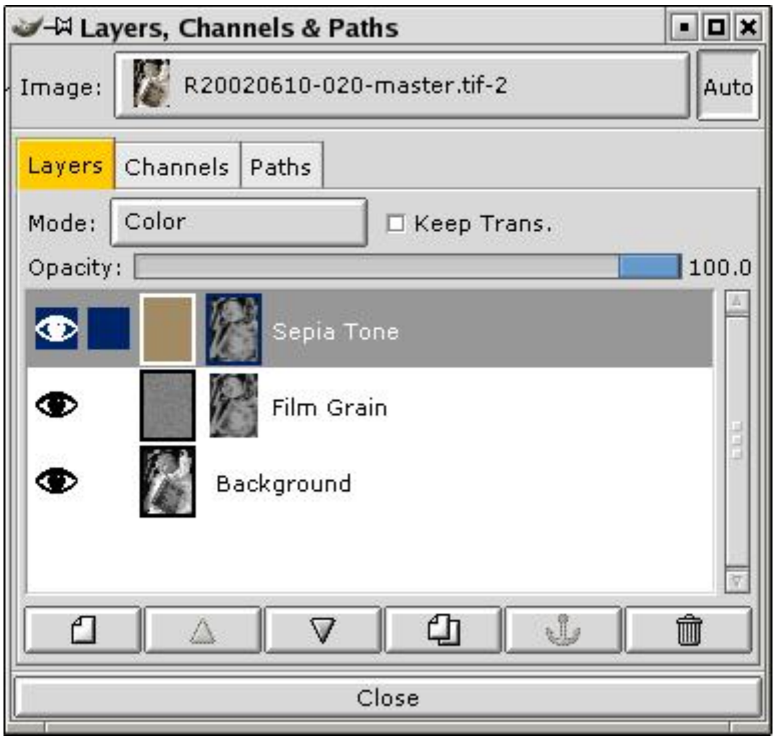
Figure 9. Black-and-White Version with Simulated Grain

Sepia Toning

Now that we have an artistic-looking black-and-white photo, let's sepia-tone it to add a more timeless and classic look. In the Layers dialog, make sure the Film Grain layer is selected, and click on the duplicate layer button at the bottom of the dialog; the entire layer is duplicated, including the layer mask. Double-click on the name Film Grain copy and change it to Sepia Tone. Change the blend mode of the new layer to Color; this applies the hue and saturation of the Sepia Tone layer with the value (luminance) of the composite layers below. Finally, click on the layer thumbnail icon in the Sepia Tone layer to select it, as opposed to its layer mask.

Click on the foreground color swatch in the main GIMP toolbox to bring up the Color Selection dialog, as you did before. Dial in the color (red = 162, green = 138 and blue = 101) and click OK. You should see the color swatch change to a hue of brown. Once you have this technique down, you can experiment with different tinting colors, but this is a good starting point.

Select the Fill tool (paint bucket) in The GIMP toolbox window and move your mouse focus to the image window. Do a select all (Ctrl-A), and click once in the window to fill it with the new color. Your Layers dialog should look something like Figure 10, and the image should take on an overall sepia tint, as shown in Figure 11.



Mode: Color
☐ Keep Trans.
Opacity: 100.0
Sepia Tone
Film Grain
Background

Figure 10. The Layers Dialog after Adding a Sepia Tone Layer

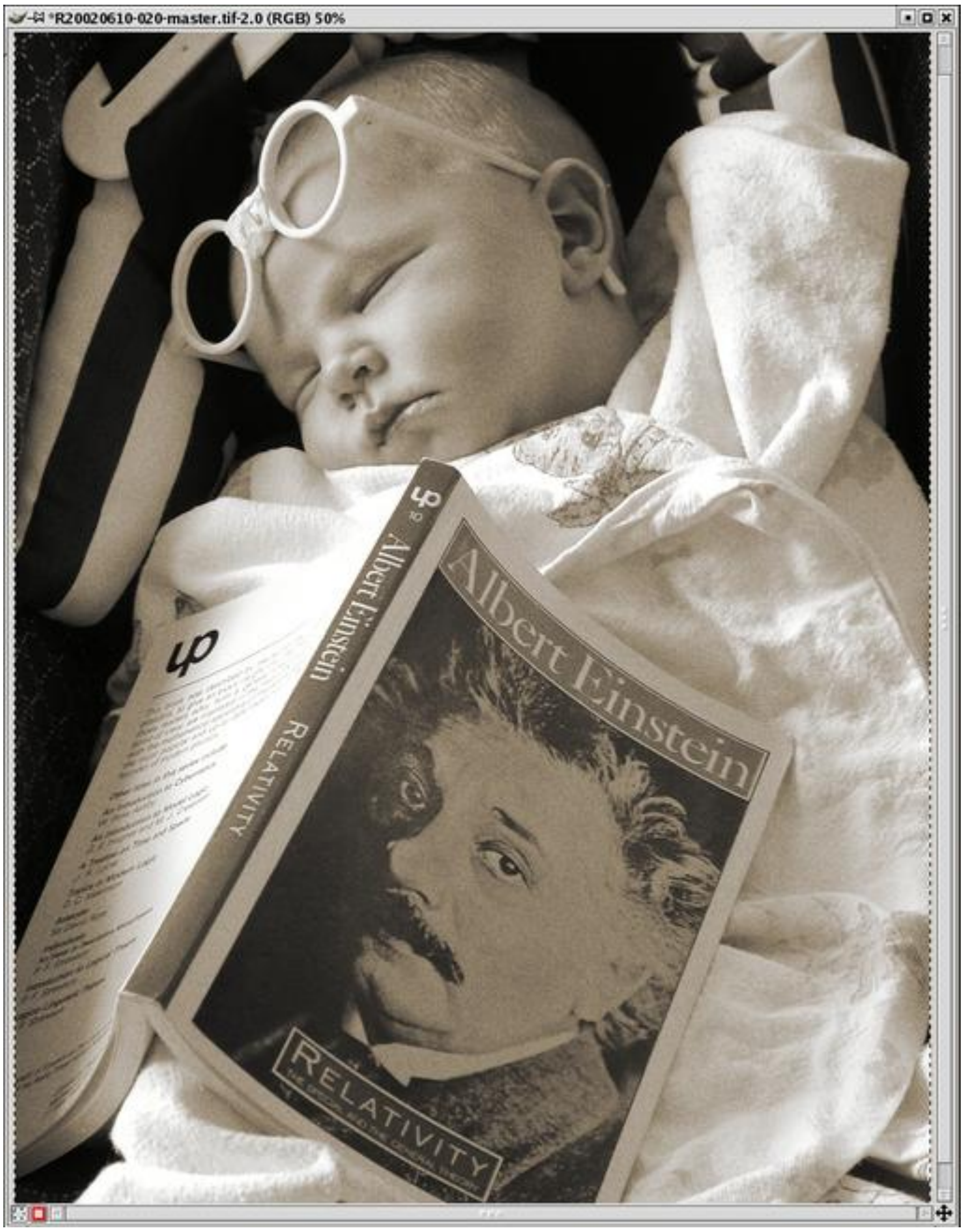


Figure 11. Grainy Photo after Sepia Toning

The use of the same layer mask in the Sepia Tone layer as the Film Grain layer is deliberate. In the traditional wet-darkroom process, the sepia toning occurs most in the mid-tones, and the lighter and darker areas appear to be less brown. You can use the same Ctrl-click trick on the layer mask to see its effect on the sepia toning. If you prefer the stronger sepia cast that you achieve without the layer mask, simply right-click on the layer and select Delete Layer Mask.

Vignetting

Time to add our final effect to make this image really stand out. We'll add a vignette, a sort of spotlight effect that draws the viewer's eye to the subject of the photograph by creating a gradual darkening moving radially away from the main subject. If a vignette is done subtly and properly, the viewer is not consciously aware of the manipulation. It is a technique particularly well suited to portraits.

In the Layers dialog, make sure that the Background layer is selected and click on the duplicate layer button at the bottom of the dialog. Double-click on the name Background copy and change it to Vignette. Right-click on the Vignette layer and add a layer mask with White (Full Opacity).

In The GIMP toolbox, click on the tiny versions of the black and white color swatches to restore the default foreground to black. Now double-click the Blend (Gradient) tool to select it and open the Tool Options dialog. In the dialog, select Radial as the Gradient type. Go back to the image window and click down in the center of the area where you want the center of the vignette spotlight to be, then drag outward toward a corner and release. In this example, I clicked in the baby's chin and dragged nearly to the upper-right corner. The only change you should notice is a radial gradient that appears in the layer mask icon, as shown in Figure 12.

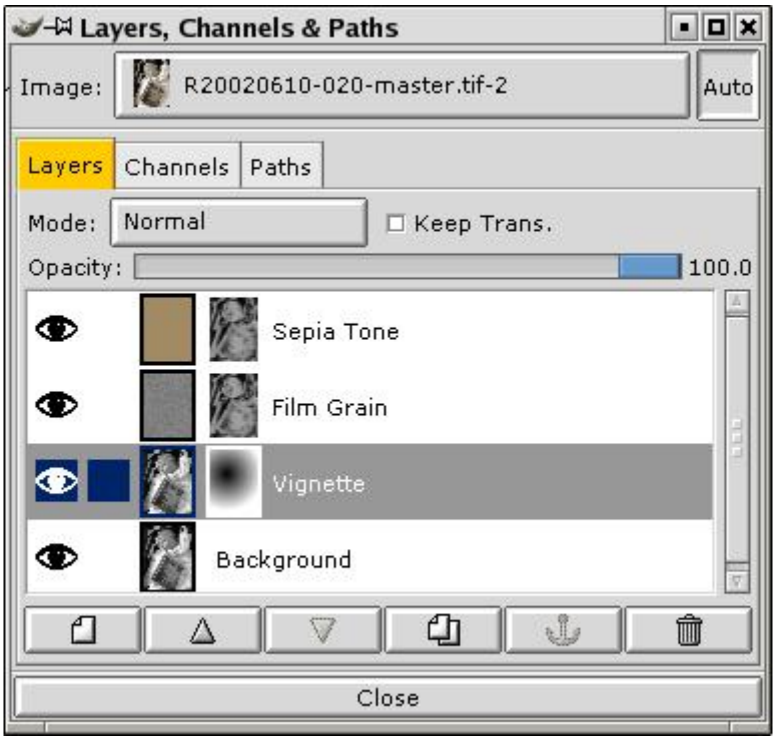


Image: R20020610-020-master.tif-2
Mode: Normal
☐ Keep Trans.
Opacity: 100.0
Sepia Tone
Film Grain
Vignette
Background

Figure 12. The Layers Dialog after Adding the Vignette Layer

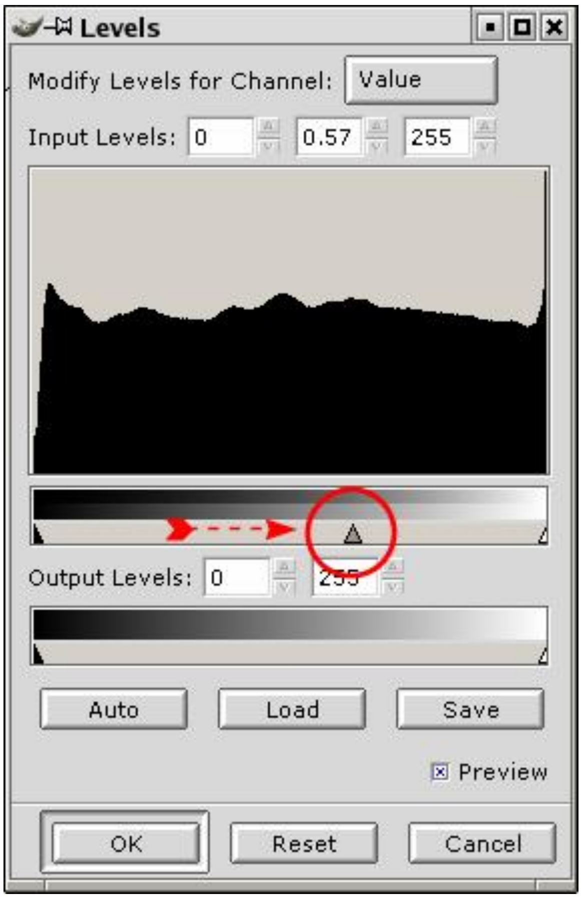


Figure 13. Adjusting Levels to Finish the Vignette Effect

Go back to the Vignette layer in the Layers dialog and click on the layer icon to select it instead of the layer mask. Now, go to the image window and bring up the Levels dialog (RC→Image→Colors→Levels). Move the middle (gray) slider to the right a bit and release it, as shown in Figure 13. Check the effect in the image window and readjust the slider until you are satisfied with the effect, then click OK. You can see my final result in Figure 14; the vignette draws the eye to the baby's face and adds a pleasing shadowy contrast overall.

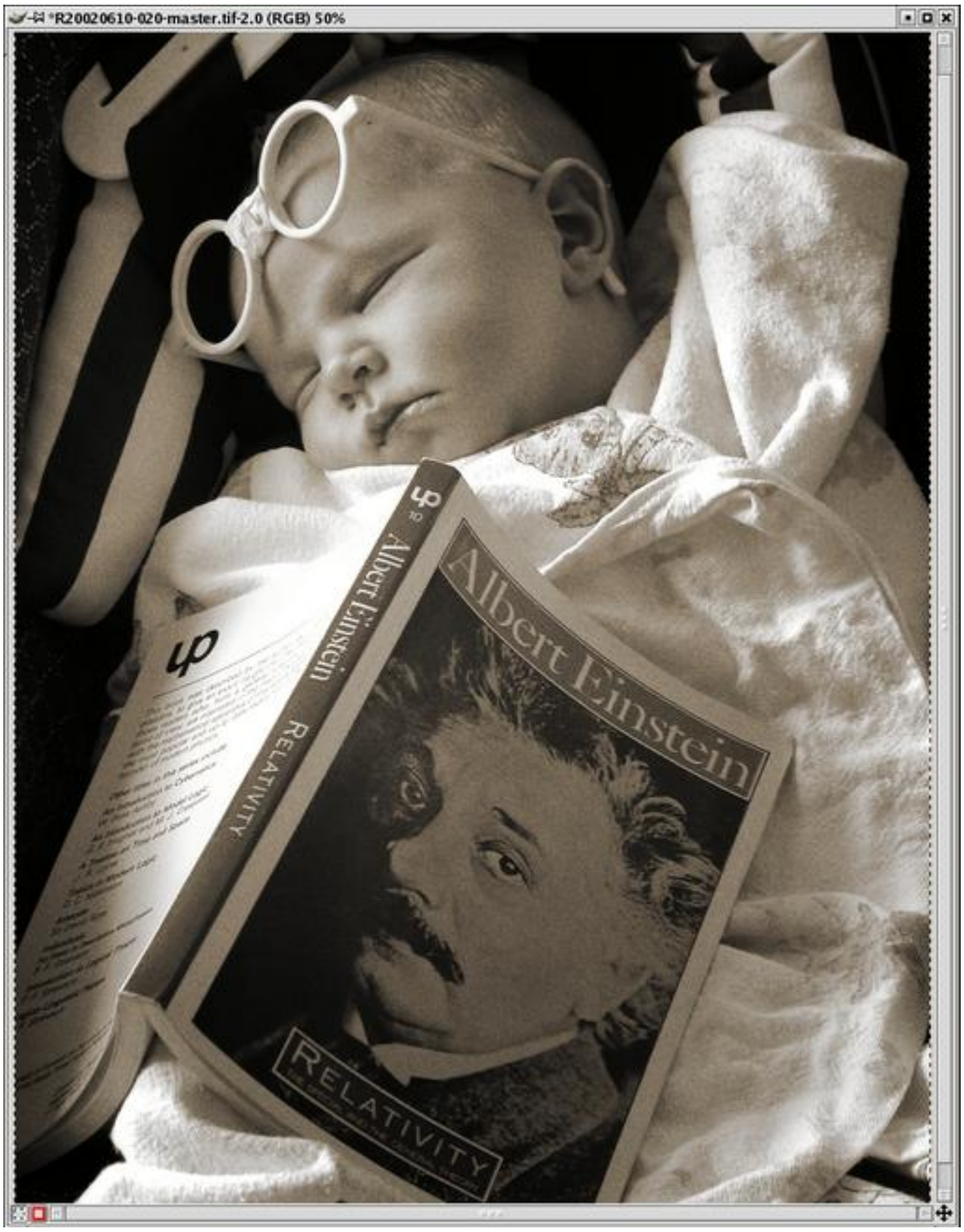


Figure 14. The Finished Image with All Effects Applied

If you need to redraw your gradient to try a different spotlight effect, simply select the Vignette layer mask icon, click the Blend tool and try again in the image window; the new gradient replaces the old.

Time to sit back and have some fun evaluating your handiwork. Try clicking the eye next to the various layers in the layers dialog to toggle their visibility on and off—an easy way to preview the different effects we have discussed. If you like the image better without one of the effects, delete that layer by selecting it and clicking the trash can icon.

If you want to do further image editing on the photo, it might be a good idea to save your work under a new name at this point or duplicate the image (Ctrl-D) and flatten it (RC→Layers→Flatten Image).

It's a good idea to give yourself a check-pointed result that you can start over with if further edits go awry. You can experiment with changing the hue and saturation, punch up the contrast with levels and curves or do any other desired edits at this point on the flattened version.

I hope I've given you a sense of the power of image layers in this article. Although a layered approach to image editing requires more memory resources, it pays off in great flexibility. So, slap some extra memory in your Linux box and build yourself a layered antique masterpiece.

Eric Jeschke (eric@redskiesatnight.com) holds a PhD in Computer Science from Indiana University and has worked as a software engineer, university professor and freelance consultant. He lives in Hawaii with his wife, kids and an overweight cat. Eric enjoys his family, outdoor adventures, taking photographs and running Linux.

Archive Index Issue Table of Contents

Advanced search

Distributed Hash Tables, Part I

Brandon Wiley

Issue #114, October 2003

Distributed hash tables are an essential component of robust peer-to-peer networks. Learn to write applications that let everyone's copy share the same data.

In the world of decentralization, distributed hash tables (DHTs) recently have had a revolutionary effect. The chaotic, ad hoc topologies of the first-generation peer-to-peer architectures have been superseded by a set of topologies with emergent order, provable properties and excellent performance. Knowledge of DHT algorithms is going to be a key ingredient in future developments of distributed applications.

A number of research DHTs have been developed by universities, picked up by the Open Source community and implemented. A few proprietary implementations exist as well, but currently none are available as SDKs; rather, they are embedded in commercially available products. Each DHT scheme generally is pitched as being an entity unto itself, different from all other schemes. In actuality, the various available schemes all fit into a multidimensional matrix. Take one, make a few tweaks and you end up with one of the other ones. Existing research DHTs, such as Chord, Kademlia and Pastry, therefore are starting points for the development of your own custom schemes. Each has properties that can be combined in a multitude of ways. In order to fully express the spectrum of options, let's start with a basic design and then add complexity in order to gain useful properties.

Basically, a DHT performs the functions of a hash table. You can store a key and value pair, and you can look up a value if you have the key. Values are not necessarily persisted on disk, although you certainly could base a DHT on top of a persistent hash table, such as Berkeley DB; and in fact, this has been done. The interesting thing about DHTs is that storage and lookups are distributed among multiple machines. Unlike existing master/slave database replication architectures, all nodes are peers that can join and leave the network freely.

Despite the apparent chaos of periodic random changes to the membership of the network, DHTs make provable guarantees about performance.

To begin our exploration of DHT designs, we start with a circular, double-linked list. Each node in the list is a machine on the network. Each node keeps a reference to the next and previous nodes in the list, the addresses of other machines. We must define an ordering so we can determine what the “next” node is for each node in the list. The method used by the Chord DHT to determine the next node is as follows: assign a unique random ID of k bits to each node. Arrange the nodes in a ring so the IDs are in increasing order clockwise around the ring. For each node, the next node is the one that is the smallest distance clockwise away. For most nodes, this is the node whose ID is closest to but still greater than the current node's ID. The one exception is the node with the greatest ID, whose successor is the node with the smallest ID. This distance metric is defined more concretely in the distance method (Listing 1).

Listing 1. ringDistance.py

```
# This is a clockwise ring distance function.
# It depends on a globally defined k, the key size.
# The largest possible node id is 2**k.
def distance(a, b):
    if a==b:
        return 0
    elif a<b:
        return b-a;
    else:
        return (2**k)+(b-a);
```

Each node is itself a standard hash table. All you need to do to store or retrieve a value from the hash table is find the appropriate node in the network, then do a normal hash table store or lookup there. A simple way to determine which node is appropriate for a particular key (the one Chord uses) is the same as the method for determining the successor of a particular node ID. First, take the key and hash it to generate a key of exactly k bits. Treat this number as a node ID, and determine which node is its successor by starting at any point in the ring and working clockwise until a node is found whose ID is closest to but still greater than the key. The node you find is the node responsible for storage and lookup for that particular key (Listing 2). Using a hash to generate the key is beneficial because hashes generally are distributed evenly, and different keys are distributed evenly across all of the nodes in the network.

Listing 2. findNode.py

```
# From the start node, find the node responsible
# for the target key
def findNode(start, key):
    current=start
    while distance(current.id, key) > \
          distance(current.next.id, key):
        current=current.next
    return current

# Find the responsible node and get the value for
# the key
def lookup(start, key):
    node=findNode(start, key)
    return node.data[key]

# Find the responsible node and store the value
# with the key
def store(start, key, value):
    node=findNode(start, key)
    node.data[key]=value
```

This DHT design is simple but entirely sufficient to serve the purpose of a distributed hash table. Given a static network of nodes with perfect uptime, you can start with any node and key and find the node responsible for that key. An important thing to keep in mind is that although the example code so far looks like a fairly normal, doubly linked list, this is only a simulation of a DHT. In a real DHT, each node would be on a different machine, and all calls to them would need to be communicated over some kind of socket protocol.

In order to make the current design more useful, it would be nice to account for nodes joining and leaving the network, either intentionally or in the case of failure. To enable this feature, we must establish a join/leave protocol for our network. The first step in the Chord join protocol is to look up the successor of the new node's ID using the normal lookup protocol. The new node should be inserted between the found successor node and that node's predecessor. The new node is responsible for some portion of the keys for which the predecessor node was responsible. In order to ensure that all lookups work without fail, the appropriate portion of keys should be copied to the new node before the predecessor node changes its next node pointer to point to the new node.

Leaves are very simple; the leaving node copies all of its stored information to its predecessor. The predecessor then changes its next node pointer to point to the leaving node's successor. The join and leave code is similar to the code for inserting and removing elements from a normal linked list, with the added requirement of migrating data between the joining/leaving nodes and their neighbors. In a normal linked list, you remove a node to delete the data it's holding. In a DHT, the insertion and removal of nodes is independent of the insertion and removal of data. It might be useful to think of DHT nodes as similar to the periodically readjusting buckets used in persistent hash table implementations, such as Berkeley DB.

Allowing the network to have dynamic members while ensuring that storage and lookups still function properly certainly is an improvement to our design. However, the performance is terrible—O(n) with an expected performance of n/2. Each node traversed requires communication with a different machine on the network, which might require the establishment of a TCP/IP connection, depending on the chosen transport. Therefore, n/2 traversed nodes can be quite slow.

In order to achieve better performance, the Chord design adds a layer to access O(log n) performance. Instead of storing a pointer to the next node, each node stores a “finger table” containing the addresses of k nodes. The distance between the current node's ID and the IDs of the nodes in the finger table increases exponentially. Each traversed node on the path to a particular key is closer logarithmically than the last, with O(log n) nodes being traversed overall.

In order for logarithmic lookups to work, the finger table needs to be kept up to date. An out-of-date finger table doesn't break lookups as long as each node has an up-to-date next pointer, but lookups are logarithmic only if the finger table is correct. Updating the finger table requires that a node address is found for each of the k slots in the table. For any slot x, where x is 1 to k, finger[x] is determined by taking the current node's ID and looking up the node responsible for the key $(id+2^{(x-1)}) \bmod (2^k)$ (Listing 3). When doing lookups, you now have k nodes to choose from at each hop, instead of only one at each. For each node you visit from the starting node, you follow the entry in the finger table that has the shortest distance to the key (Listing 4).

Listing 3. update.py

```
def update(node):
    for x in range(k):
        oldEntry=node.finger[x]
        node.finger[x]=findNode(oldEntry,
                          (node.id+(2**x)) % (2**k))
```

Listing 4. finger-lookup.py

```
def findFinger(node, key):
    current=node
    for x in range(k):
        if distance(current.id, key) > \
           distance(node.finger[x].id, key):
            current=node.finger[x]
    return current

def lookup(start, key):
    current=findFinger(start, key)
    next=findFinger(current, key)
    while distance(current.id, key) > \
          distance(next.id, key):
        current=next
        next=findFinger(current, key)
    return current
```

So far we have more or less defined the original version of the Chord DHT design as it was described by the MIT team that invented it. This is only the tip of the iceberg in the world of DHTs, though. Many modifications can be made to establish different properties from the ones described in the original Chord paper, without losing the logarithmic performance and guaranteed lookups that Chord provides.

One property that might be useful for a DHT is the ability to update the finger table passively, requiring periodic lookups to be done in order to refresh the table. With MIT Chord, you must do a lookup, hitting O(log n) nodes for all k items in the finger table, which can result in a considerable amount of traffic. It would be advantageous if a node could add other nodes to its finger table when they contacted it for lookups. As a conversation already has been established in order to do the lookup, there is little added overhead in checking to see if the node doing the lookup is a good candidate for the local finger table. Unfortunately, finger table links in Chord are unidirectional because the distance metric is not symmetrical. A node generally is not going to be in the finger tables of the nodes in its finger table.

A solution to this problem is to replace Chord's modular addition distance metric with one based on XOR. The distance between two nodes, A and B, is defined as the XOR of the node IDs interpreted as the binary representation of an unsigned integer (Listing 5). XOR makes a delightful distance metric because it is symmetric. Because distance(A, B) == distance(B, A), for any two nodes, if A is in B's finger table then B is in A's finger table. This means nodes can update their finger tables by recording the addresses of nodes that query them, reducing significantly the amount of node update traffic. It also simplifies coding a DHT application, because you don't need to keep a separate thread to call the update method periodically. Instead, you simply update whenever the lookup method is called.

Listing 5. xor-distance.py

```
def distance(a, b):
    return a^b # In Python, this means a XOR b,
               # not a to the power of b.
```

An issue with the design presented so far is the paths to a given node are fragile. If any node in the path refuses to cooperate, the lookup is stuck. Between any two nodes there is exactly one path, so routing around broken nodes is impossible. The Kademlia DHT solves this by widening the finger table to contain a bucket of j references for each finger table slot instead of only one, where j is defined globally for the whole network. Now j different choices are available for each hop, so there are somewhere around j*log(n) possible paths. There are less than that, though, paths converge as they get closer to the target. But, the number of possible paths probably is greater than 1, so this is an improvement.

Kademlia goes further and orders the nodes in the bucket in terms of recorded uptime. Older nodes are given preference for queries, and new references are added only if there are not enough old nodes. Besides the increased reliability of queries, this approach offers the added benefit that an attack on the network in which new nodes are created rapidly in order to push out good nodes will fail—it won't even be noticeable.

It's important to understand that these different properties are not tied to a particular DHT implementation. We gradually have built up a DHT design from scratch, developed it into something that resembles Chord, then modified it to be more like Kademlia. The different approaches can be more or less mixed and matched. Your finger table buckets can have 1 slot or j slots, depending on whether you use modular addition or XOR for your distance metric. You can always follow the closest node, or you can rank them according to uptime or according to some other criteria. You can draw from several other DHT designs, such as Pastry, OceanStore and Coral. You also can use your own ideas to devise the perfect design for your needs. Myself, I have concocted several modifications to a base Chord design to add properties such as anonymity, Byzantine fault-tolerant lookups, geographic routing and the efficient broadcasting of messages to enter the network. It's fun to do and easier than you think.

Now that you know how to create your own DHT implementations, you're probably wondering what kind of crazy things you can do with this code. Although there probably are many applications for DHTs that I haven't thought of yet, I know people already are working on such projects as file sharing, creating a shared hard drive for backing up data, replacing DNS with a peer-to-peer name resolution system, scalable chat and serverless gaming.

For this article, I've tied the code together into a fun little example application that might be of interest to readers who caught my interview on the *Linux Journal* Web site about peer-to-peer Superworms (see Resources). The application is a distributed port scanner that stores results in the simulated DHT (Listing 6). Given a fully functional DHT implementation, this script would have some handy properties. First, it allows multiple machines to contribute results to a massive scanning of the Internet. This way, all of the scanning activity can't be linked with a single machine. Additionally, it avoids redundant scanning. If the host already has been scanned, the results are fetched from the DHT, avoiding multiple scans. No central server is required to hold all of the results or to coordinate the activities of the participants. This application may seem somewhat insidious, but the point is it was trivial to write given the DHT library. The same approach can be used in other sorts of distributed projects.

Listing 6. portscan.py

```
def __main__():
    id=int(random.uniform(0,2**k))
    node=Node(id)
    join(node, initialContact)

    line=raw_input('Enter an IP to scan: ').trim()
    key=long(sha.new(line).hexdigest(),16)
    value=lookup(node, key)
    if value==None:
        f=os.popen('nmap '+args[1])
        lines=f.readlines()
        value=string.join(lines, '\n')
        store(node, key, value)
```

In this installment of our two-part series, we discussed the theory behind building DHTs. Next time, we'll talk about practical issues in using DHTs in real-world applications.

Resources

Achord: thalassocracy.org/achord/achord-iptps.html

Chord: www.pdos.lcs.mit.edu/chord

Curious Yellow: blanu.net/curious_yellow.html

How Can You Defend against a Superworm? linuxjournal.com/article/6069

Kademlia: kademlia.scs.cs.nyu.edu

Brandon Wiley is a peer-to-peer hacker and current president of the Foundation for Decentralization Research, a nonprofit corporation empowering people through technology. He can be contacted at blanu@decentralize.org.

Xilinx FPGA Design Tools for Linux

Michael Baxter

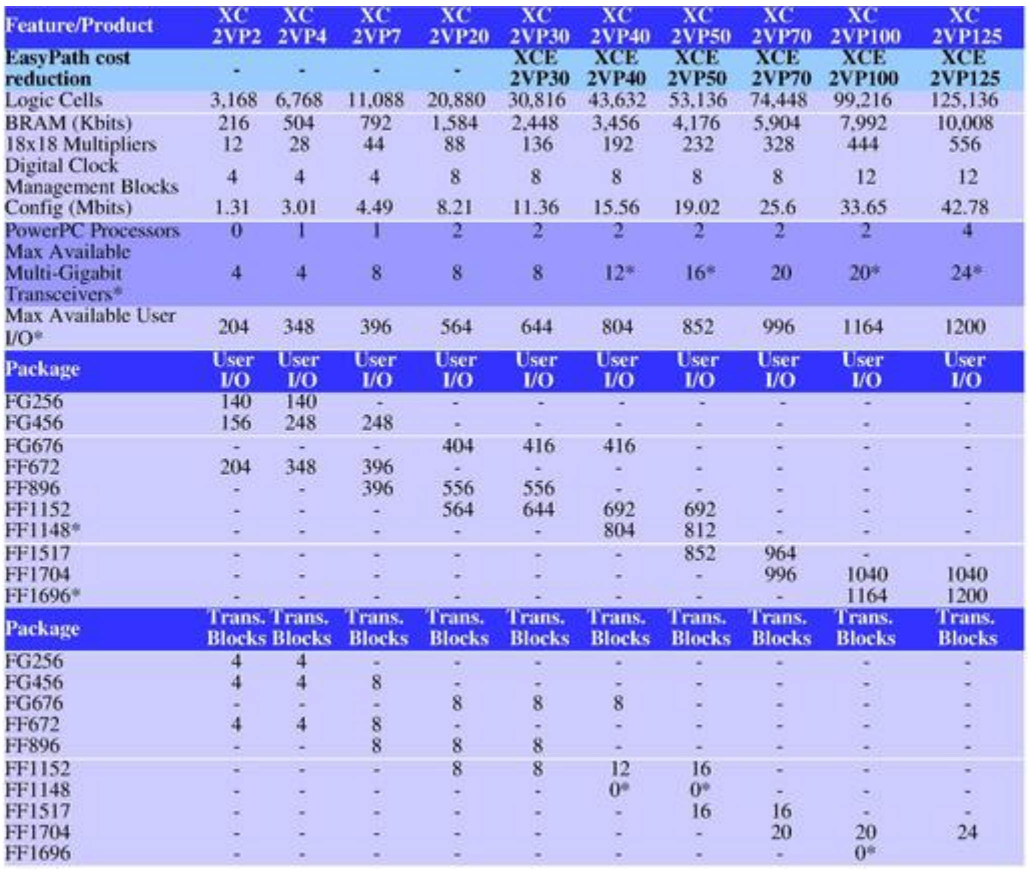
Issue #114, October 2003

Linux is making big moves into electronic design automation. Michael introduces a way to implement your designs in reconfigurable hardware.

A field programmable gate array (FPGA) is a user-programmable piece of silicon constructed in very large-scale integration (VLSI) technology. The VLSI transistor-level detail is absolutely predefined in an FPGA. Internally, the FPGA consists of a matrix-like fabric of logic and interconnect elements that are inherently flexible. Flexibility is accomplished through programmable SRAM memory cells that define the silicon resources. FPGAs are standard commodity parts with trillions of possible user configurations. This essential organizational structure of the FPGA has persisted through two decades of VLSI technology development. However, today's FPGAs are utterly unlike those of yesteryear.

FPGAs are blurring the lines between hardware and software in systems. FPGA devices are inherently soft-programmable and may be changed dynamically during the operation of a system. More compellingly, FPGA devices now also contain embedded microprocessors within the logic fabric, and these microprocessors can run Linux. Imagine a Linux computer with up to millions of gates of flexible logic immediately around it. One way to grok this new paradigm is to think of the following: “Software is configuration bits for hardware.”

FPGA design is custom silicon design with less effort than full-custom VLSI design. Besides processor cores, FPGAs today not only have logic gates and flip-flops, they also have large Block RAMs, embedded hardware multipliers, arithmetic acceleration logic, digital clock managers (DCMs) for frequency synthesis, multistandard system I/O cells with programmable line termination and multi-gigabit transceivers (MGTs). These system-oriented resources, along with the kinds of device packages and user I/O counts, are enumerated in Figure 1, which shows the most advanced of the FPGA devices, the Xilinx Virtex-II Pro family.



Feature/Product	XC 2VP2	XC 2VP4	XC 2VP7	XC 2VP20	XC 2VP30	XC 2VP40	XC 2VP50	XC 2VP70	XC 2VP100	XC 2VP125
EasyPath cost reduction	-	-	-	-	**XCE 2VP30**	**XCE 2VP40**	**XCE 2VP50**	**XCE 2VP70**	**XCE 2VP100**	**XCE 2VP125**
Logic Cells	3,168	6,768	11,088	20,880	30,816	43,632	53,136	74,448	99,216	125,136
BRAM (Kbits)	216	504	792	1,584	2,448	3,456	4,176	5,904	7,992	10,008
18x18 Multipliers	12	28	44	88	136	192	232	328	444	556
Digital Clock Management Blocks	4	4	4	8	8	8	8	8	12	12
Config (Mbits)	1.31	3.01	4.49	8.21	11.36	15.56	19.02	25.6	33.65	42.78
PowerPC Processors	0	1	1	2	2	2	2	2	2	4
Max Available Multi-Gigabit Transceivers*	4	4	8	8	8	12*	16*	20	20*	24*
Max Available User I/O*	204	348	396	564	644	804	852	996	1164	1200
Package	**User I/O**	**User I/O**	**User I/O**	**User I/O**	**User I/O**	**User I/O**	**User I/O**	**User I/O**	**User I/O**	**User I/O**
FG256	140	140	-	-	-	-	-	-	-	-
FG456	156	248	248	-	-	-	-	-	-	-
FG676	-	-	-	404	416	416	-	-	-	-
FF672	204	348	396	-	-	-	-	-	-	-
FF896	-	-	396	556	556	-	-	-	-	-
FF1152	-	-	-	564	644	692	692	-	-	-
FF1148*	-	-	-	-	-	804	812	-	-	-
FF1517	-	-	-	-	-	-	852	964	-	-
FF1704	-	-	-	-	-	-	-	996	1040	1040
FF1696*	-	-	-	-	-	-	-	-	1164	1200
Package	**Trans. Blocks**	**Trans. Blocks**	**Trans. Blocks**	**Trans. Blocks**	**Trans. Blocks**	**Trans. Blocks**	**Trans. Blocks**	**Trans. Blocks**	**Trans. Blocks**	**Trans. Blocks**
FG256	4	4	-	-	-	-	-	-	-	-
FG456	4	4	8	-	-	-	-	-	-	-
FG676	-	-	-	8	8	8	-	-	-	-
FF672	4	4	8	-	-	-	-	-	-	-
FF896	-	-	8	8	8	-	-	-	-	-
FF1152	-	-	-	8	8	12	16	-	-	-
FF1148	-	-	-	-	-	0*	0*	-	-	-
FF1517	-	-	-	-	-	-	16	16	-	-
FF1704	-	-	-	-	-	-	-	20	20	24
FF1696	-	-	-	-	-	-	-	-	0*	

Figure 1. Virtex-II Pro FPGA Family

The technology of VLSI memory is everything in this equation—FPGAs are in the realm of commodity silicon manufacturing and typically have better silicon wafer yields than custom VLSIs. Following Moore's Law, FPGAs, much like DRAM and other advanced memory products, are the lead silicon technology drivers, pushing the most advanced 300mm wafer technology at deep submicron densities.

Tools for a New Paradigm

In this article, we introduce the most recent Xilinx FPGA design tools. The design tools are called the Integrated Software Environment, or ISE. These design tools are now released for the Linux platform as the Xilinx ISE 6.1i tools for Linux. This allows FPGA design on a platform having very low total cost of ownership.

Designing with an FPGA device is both different from and similar to programming microprocessors in a language such as C. Hardware description languages (HDLs) are used to design logic at a high level. Verilog and VHDL are the most popular HDLs in industry practice, and ISE 6.1i supports both. These languages allow description of hardware in structural or behavioral terms, or as a mix of both. HDLs are the input source code for specialized compilers, which either synthesize logic for a target or allow it to be simulated. Here, our focus is on logic synthesis, with an inside look at the FPGA and, finally, generation of the configuration bitstream with the ISE 6.1i tools.

One way logic is different from software is that it's inherently parallel. HDLs can describe numerous *concurrent* changes directly, unlike the major programming languages, for example, when specifying synchronous changes in a logic circuit based on the rising edge of a clock signal. In logic design, as contrasted with programming, one is often describing something that takes *area*, not memory. Both logic and programs require that time elapses during operation, and our preference is normally that it be very little time. As we will see, the ISE 6.1i design tools can help with both area and time optimization.

This article covers the basic steps in entering a simple but interesting design in Verilog and explores some of the capabilities of the tool. We also look inside the FPGA device configuration. See Resources for more information about the Verilog HDL. Additionally, refer to the July 2002 issue of *LJ* (/article/6001) for an article on a free Verilog tool, which also contains a Verilog tutorial.

Setting Up a New Project with ISE 6.1i

For this tour, we synthesize a 16-bit pipelined parallel multiplier by specifying it with behavioral Verilog. We use synthesis to create and evaluate this result. Then, we use the implementation tools in ISE 6.1i to create a configuration for a particular Virtex-II Pro FPGA device. Various options of the toolchain are explained in the process, including a way to look inside the implemented FPGA.

After the software is installed, we open a shell and begin. The process starts by typing `ise` at the command line. This brings up the Project Navigator, which is the screen shown in Figure 2. To start a new FPGA design project, select a new project under the File menu. This brings up the window shown in Figure 3. Here, we enter a project name, MPY-TEST, and indicate the kind of top-level module we're going to use for this project. We are interested in an HDL top-level module for this tour, but ISE 6.1i allows the use of several other top-level module types.

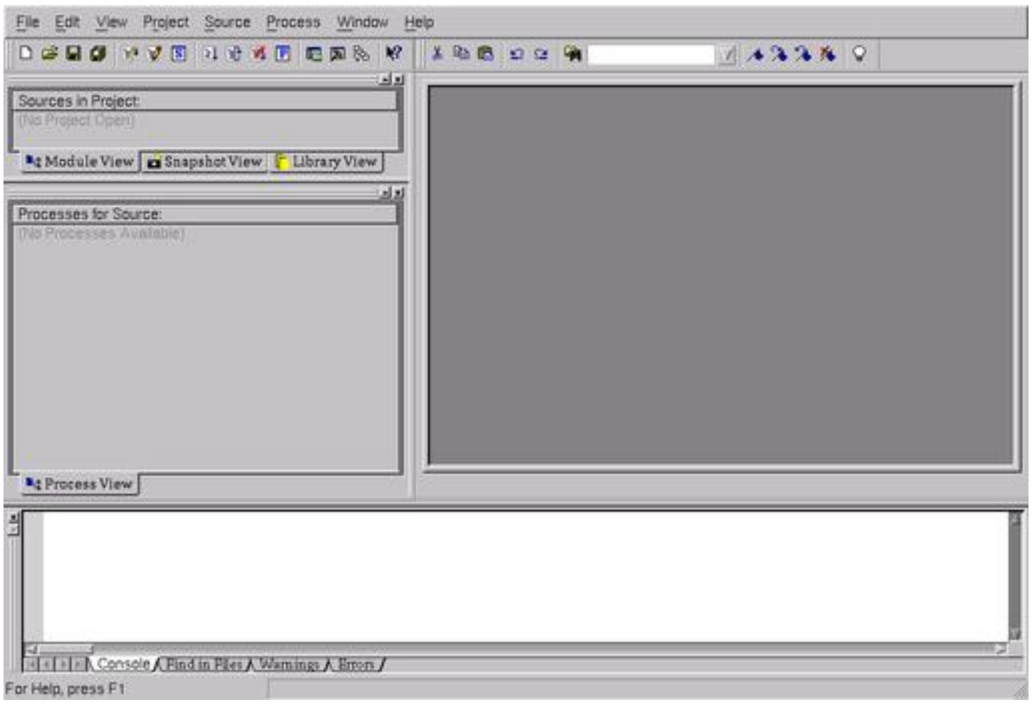
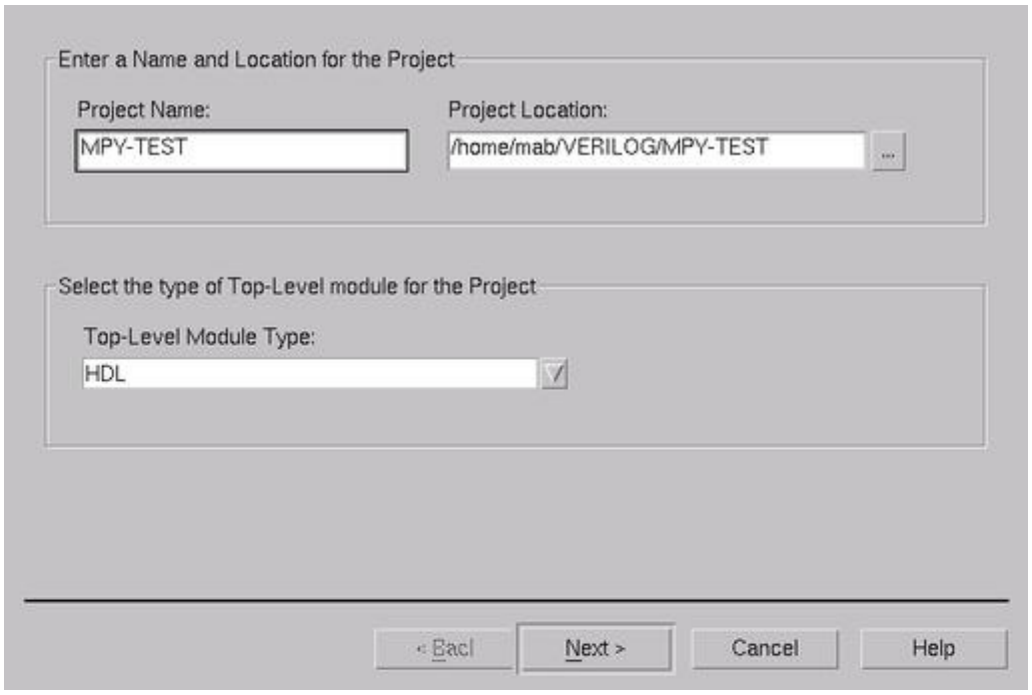


Figure 2. ISE 6.1i Startup Screen


Enter a Name and Location for the Project

Project Name: MPY-TEST

Project Location: /home/mab/VERILOG/MPY-TEST

Select the type of Top-Level module for the Project

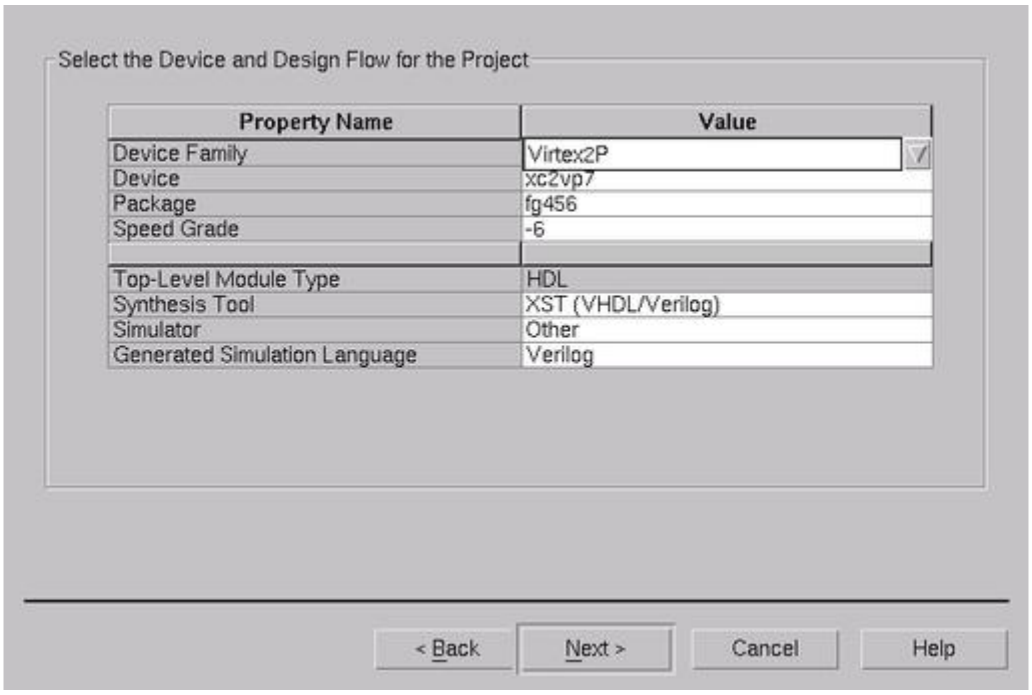
Top-Level Module Type: HDL

Figure 3. Entering an FPGA New Project

Selecting an FPGA Device

Now that we've selected HDL, the next step is to let the tool know with which kind of FPGA we're going to design. For illustration, we use the most advanced FPGA devices, the Virtex-II Pro family. This is shown in Figure 4, where the xc2vp7 device is selected, along with an fg456 package. This device is one of the smaller Virtex-II Pro parts, but it still has numerous resources. Here, we use only a small part of the device.

The menu window in Figure 4 also allows other choices. You can select the part speed grade, and here we accept the default of -6. The Project Navigator can be used to organize your entire project flow. For instance, you can perform both behavioral simulation and functional simulation. The first type of simulation is a check that you have a logically correct design—that the design does what it's supposed to do. The second type of simulation is post-FPGA implementation, used for design verification of a completed chip. The device selection screen in Figure 4 also includes other options, such as the particular simulator you want to use for HDL simulation and what language to use for simulating an implemented FPGA. This might include, respectively, some industry-standard tools provided by Xilinx partners, or another simulator, and Verilog or VHDL.


Select the Device and Design Flow for the Project

Property Name	Value
Device Family	Virtex2P
Device	xc2vp7
Package	fg456
Speed Grade	-6
Top-Level Module Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	Other
Generated Simulation Language	Verilog

Figure 4. Setting FPGA Device Type

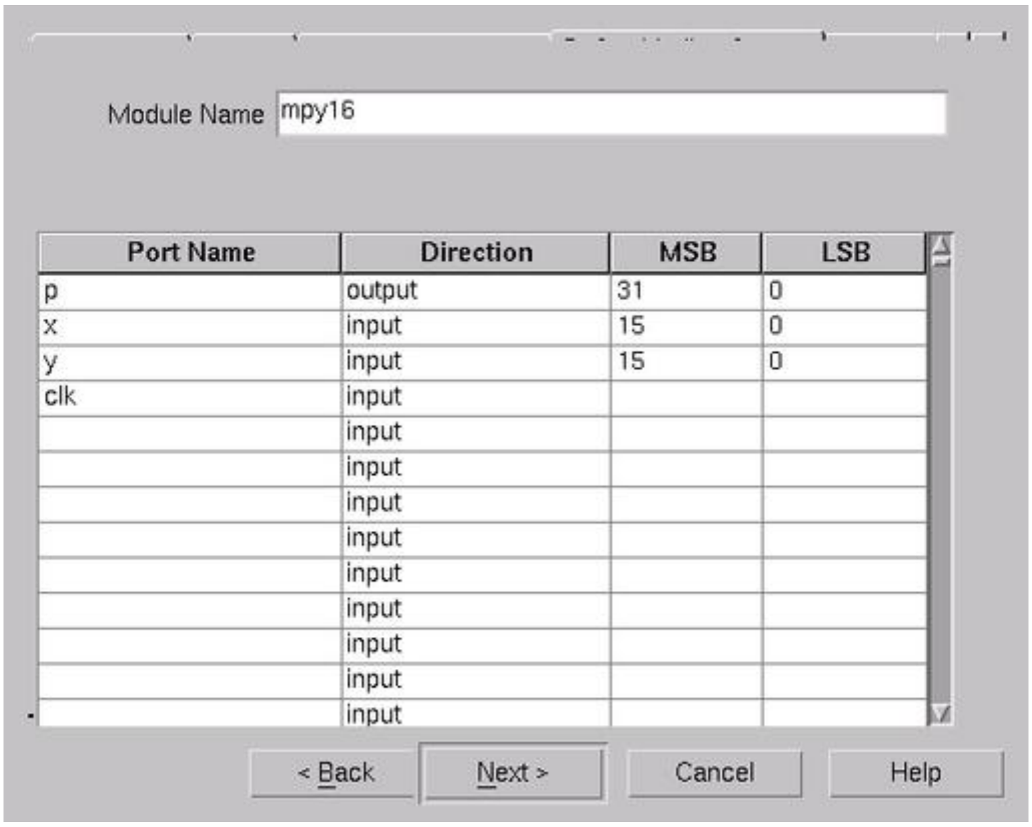
Entering a Design in Verilog

Next, we select a new source file for the design and give the design a filename. In this process, we tell the tool what kind of CAD document we're creating; in this case, it's a Verilog module. We enter a filename of mpy16.v, with the .v being the standard filename suffix for Verilog. It is customary, but not required to make the filename for the top-level module the same as the module, or a name like toplevel.v.

Several other kinds of documents can be entered for the tool and added to the project. We don't have time to examine all of these capabilities, which include alternative entry modes (schematics) and the inclusion of standard and custom HDL libraries made by the user.

To define this (first) Verilog source for the design, the Design Manager offers some help. For the top-level module mpy16, we fill out a module port table using a tabular entry tool (Figure 5). Here, we define the wires that enter and exit the top-level module, and these will end up as the external I/O pins on the FPGA. The names of the ports entered are p, x, y and clk.

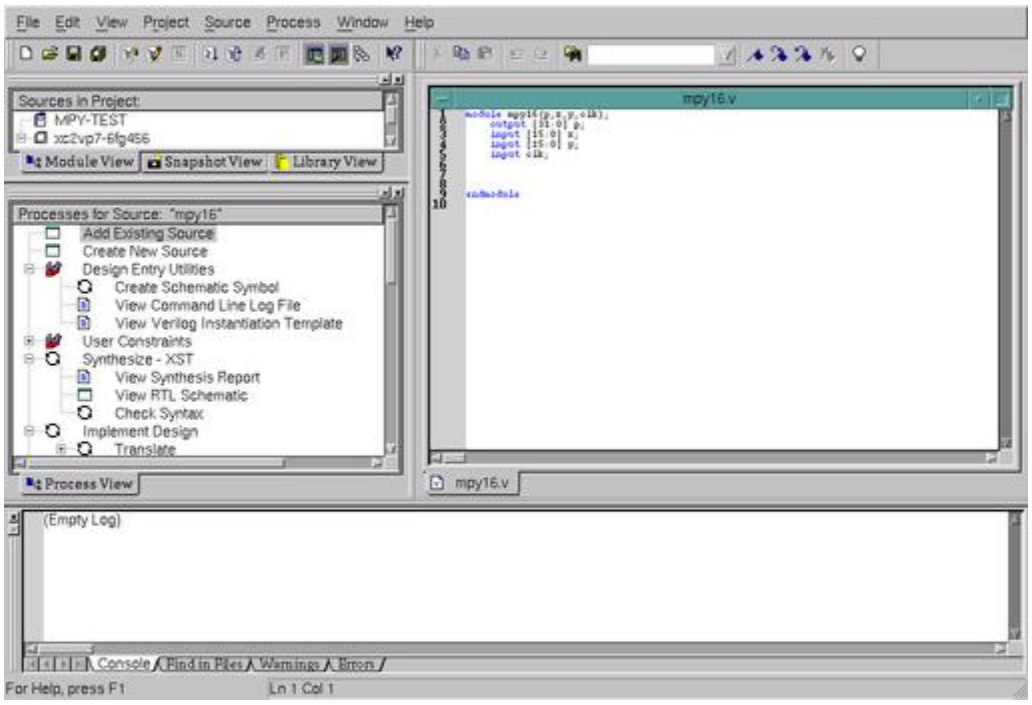
We specify p as a 32-bit-wide output and the main inputs, x and y, as 16-bits wide. Because this multiplier will be pipelined, we also include a port named clk, which will provide the synchronous timing source for the multiplier. The port clk is only a single wire, or net, so we leave out content for MSB or LSB in the table. This means clk will be a scalar. In Verilog, vectors are groups of wires or nets, and these are zero-base indexed.


Module Name mpy16

Port Name	Direction	MSB	LSB
p	output	31	0
x	input	15	0
y	input	15	0
clk	input		
	input		
	input		
	input		
	input		
	input		
	input		
	input		
	input		
	input		

Figure 5. Setting Up the External I/O Pins

After completing tabular entry for the top-level module, we obtain a summary dialog. Then with the project set up, the Project Navigator brings up all the tools and the initial outline for our Verilog module. This is shown in Figure 6, with the skeleton source code in the upper-right corner. An editor is supplied with ISE 6.1i, and you also can import HDL source code created with the Linux editor of your choice.



```
module mpy16(p,x,y,clk);
    output [31:0] p;
    input [15:0] x;
    input [15:0] y;
    input clk;



endmodule
```

Figure 6. Project, Tools and Skeleton Source

Listing 1. The Verilog Source Code for a 16-Bit Pipelined Multiplier

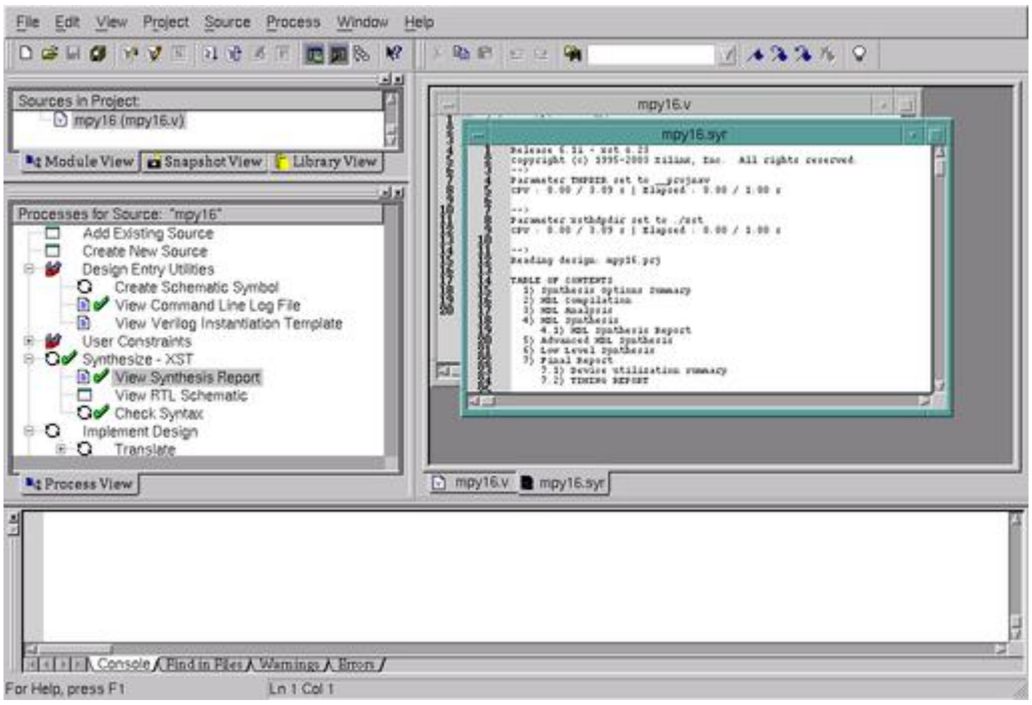
```
module mpy16(p,x,y,clk);
        output [31:0] p;
        input [15:0] x;
        input [15:0] y;
        input clk;

        // inferable storage via synthesis
        reg [31:0] p;
        reg [15:0] xq;
        reg [15:0] yq;

        // 16x16 unsigned multiplier specified
        // behaviorally
        always @(posedge clk)
                begin
                        xq <= x;
                        yq <= y;
                        p <= xq * yq;
                end
endmodule // mpy16
```

Listing 1 is the Verilog source code for a 16-bit pipelined multiplier. This code is done in a behavioral style, and we're going to allow Xilinx Synthesis Technology (XST) to figure how to implement what we mean by the code. Today, synthesis is very powerful, and we simply can infer the multiplier hardware, without having to specify its logic design in detail.

FPGA Hardware Synthesis

With the Verilog entered, as shown in Listing 1, we start synthesis by double-clicking on the Synthesize - XST button in the Project Navigator tool scroll list. This design we've entered is easy to synthesize and takes only a few moments.

The synthesis results can be viewed by reading the synthesis report file (Figure 7). The check marks in the tool scroll list at the left indicate what Project Navigator accomplished. The synthesis report file is in the upper right. You can look through this to see how XST decided to infer your logic to the target FPGA, as well as glean an estimated clock-rate performance for the resulting design in that FPGA.



```
Release 6.1i - xst G.23
Copyright (c) 1995-2003 Xilinx, Inc.  All rights reserved.
-->
Parameter TMPDIR set to __projnav
CPU : 0.00 / 3.09 s | Elapsed : 0.00 / 1.00 s

-->
Parameter xsthdpdir set to ./xst
CPU : 0.00 / 3.09 s | Elapsed : 0.00 / 1.00 s

-->
Reading design: mpy16.prj

TABLE OF CONTENTS
  1) Synthesis Options Summary
  2) HDL Compilation
  3) HDL Analysis
  4) HDL Synthesis
     4.1) HDL Synthesis Report
  5) Advanced HDL Synthesis
  6) Low Level Synthesis
  7) Final Report
     7.1) Device utilization summary
     7.2) TIMING REPORT
```

Figure 7. Viewing the Synthesis Report File

RTL is a view of your logic as various black boxes or logic gates that are situated between flip-flop registers. RTL is the predominant view of logic in synchronous design. The View RTL Schematic button in the ISE 6.1i tools offers an excellent way to get a graphical overview (Figure 8). You can navigate across any hierarchy of modules in your design with the RTL schematic being regenerated dynamically. We're now ready to run the FPGA implementation tools.

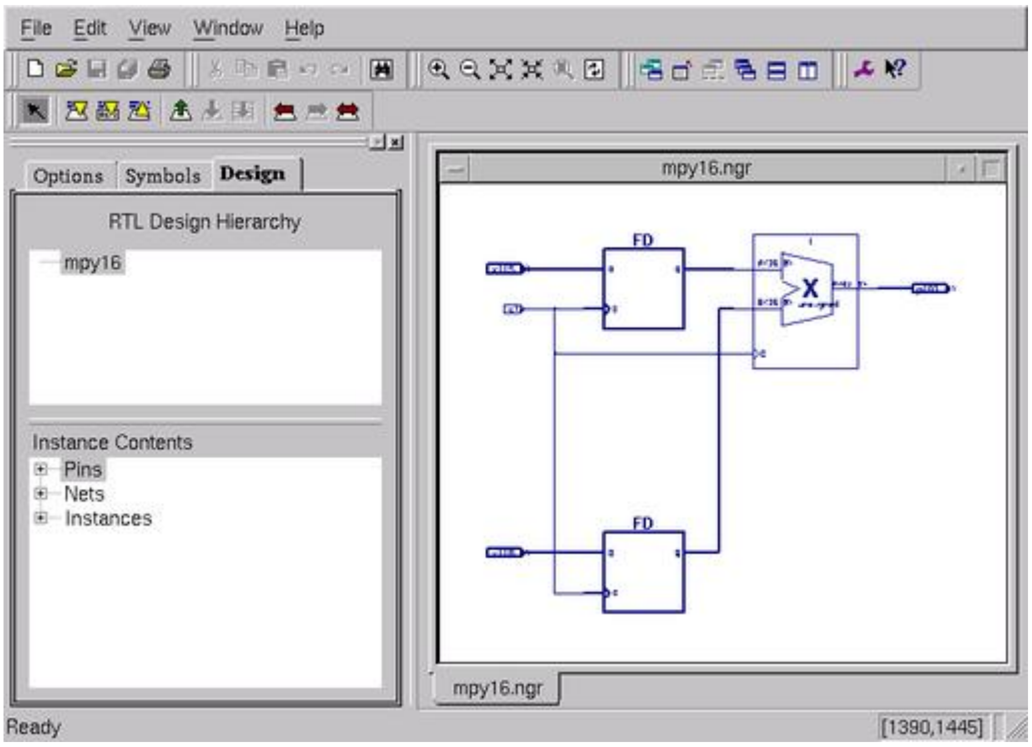


Figure 8. RTL Schematic

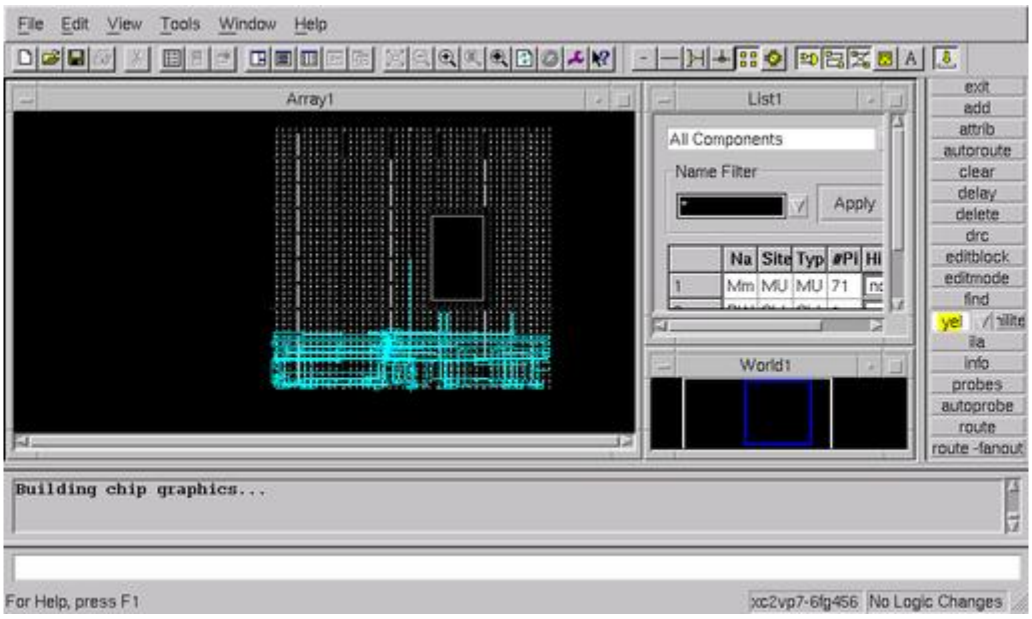
FPGA Implementation

We start implementation by clicking Implement Design in the tool list at the left in Project Navigator. For FPGA implementation, the generic process automatically engaged by the tools goes as follows: translate, map and place and route.

In most cases with FPGA design, you do not need to know the specifics of the operation of these steps, with the exception of seeking extremely high performance from the FPGA. But it can be helpful to know what's going on. For example, map determines how the inferred synthesis results get “mapped” to your selected target FPGA. Looking at the map report file can provide useful information about the allocation of resources. Similarly, sometimes it can be useful to read the place and route (PAR) report file to learn whether options are available to improve the speed of your design or to constrain resource utilization more carefully earlier in the toolchain.

Other tools in ISE 6.1i not encountered in this article provide much flexibility for performance engineering and for creating constraints such as locking the FPGA I/O pins for subsequent use with PCB boards, given iterative FPGA implementation to fix bugs.

This design used only one out of the 44 available multipliers on the xc2vp7 FPGA. Also indicated are the 65 I/O pins that were requested for this logic. The Pad report generated by the tool will tell you which FPGA pins were selected in implementation. As mentioned above, you also can preselect these pins for a board design, and this is done with a constraint editor in the tool or by creating a constraint file.



```
Building chip graphics...
```

For Help, press F1 xc2vp7-6fg456 No Logic Changes

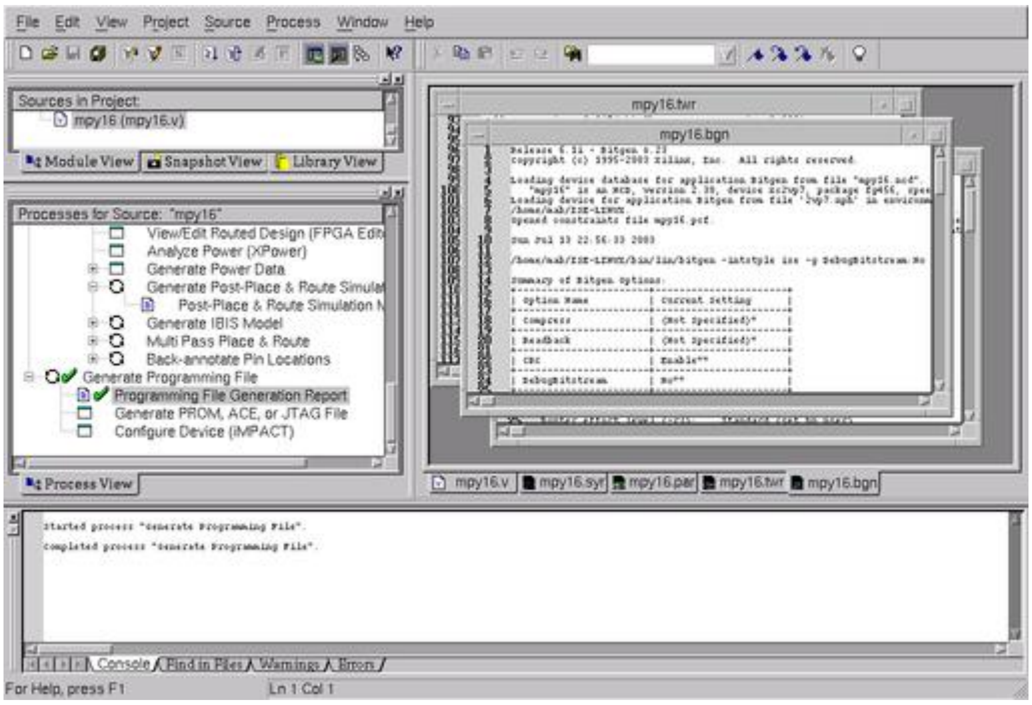
Figure 9. FPGA Worldview

FPGAs are fast CMOS devices. It's important for us to know how fast, in order to verify static timing goals against system requirements. Other important information for system design might include knowing setup time information or clock-to-pin delays. ISE 6.1i provides an integrated timing analyzer. A timing report file can be generated automatically, based on all routing and propagation delays for the completed implementation in the target FPGA.

You also can look inside the FPGA device by clicking the FPGA editor tool after implementation. This tool opens a separate screen (Figure 9). The initial view is a called a worldview, where you can see your whole FPGA at once. As you probably can see in the center view, very little of the 2vp7 was used. This view becomes quite dense with high-utilization designs. The large block in the view is the PowerPC microprocessor. Also visible are other multiplier blocks, block RAMs and numerous other elements in the FPGA. Various blocks in the design can be highlighted selectively.

Bitgen and Device Programming

The last step in FPGA implementation is bitgen, or bit generation of the configuration bitstream for the FPGA so that the device can be programmed in a system. After pressing Generate Programming File, the bitstream is created automatically from the FPGA implementation data. The bitgen tool creates a report file, which is shown in Figure 10, in the upper right.


File Edit View Project Source Process Window Help
Sources in Project:
mpy16 (mpy16.v)
Module View
Snapshot View
Library View
Processes for Source: "mpy16"
View/Edit Routed Design (FPGA Edit
Analyze Power (XPower)
Generate Power Data
Generate Post-Place & Route Simula
Post-Place & Route Simulation M
Generate IBIS Model
Multi Pass Place & Route
Back-annotate Pin Locations
Generate Programming File
Programming File Generation Report
Generate PROM, ACE, or JTAG File
Configure Device (iMPACT)
Process View
mpy16.twr
mpy16.bgn
mpy16.v
mpy16.syr
mpy16.par
mpy16.twr
mpy16.bgn
Console
Find in Files
Warnings
Errors
For Help, press F1
Ln 1 Col 1

```
Release 6.1i - Bitgen G.23
Copyright (c) 1995-2003 Xilinx, Inc.  All rights reserved.

Loading device database for application Bitgen from file "mpy16.ncd".
   "mpy16" is an NCD, version 2.38, device xc2vp7, package fg456, spe
Loading device for application Bitgen from file '2vp7.nph' in environm
/home/mab/ISE-LINUX.
Opened constraints file mpy16.pcf.

Sun Jul 13 22:56:33 2003

/home/mab/ISE-LINUX/bin/lin/bitgen -intstyle ise -g DebugBitstream:No

Summary of Bitgen Options:
+----------------------+----------------------+
| Option Name          | Current Setting      |
+----------------------+----------------------+
| Compress             | (Not Specified)*     |
+----------------------+----------------------+
| Readback             | (Not Specified)*     |
+----------------------+----------------------+
| CRC                  | Enable**             |
+----------------------+----------------------+
| DebugBitstream       | No**                 |
```

```
Started process "Generate Programming File".

Completed process "Generate Programming File".
```

Figure 10. Bitgen Results for FPGA Programming

After creating the FPGA bitstream, there are numerous options for using it to create programming files. These depend a lot on the hardware mechanism you want to use for programming the FPGA. The options include PROM file, ACE file, JTAG file and direct configuration of the FPGA.

PROM files are used for conventional PROM programmers, for example, to make an EPROM that goes in your system. ACE files are used with the Xilinx SystemACE CF programming solution. SystemACE CF is a low-cost way to embed a CompactFlash device in your system. Using ACE files, you can create data on a conventional CompactFlash card that can be used for FPGA configuration, dynamic reconfiguration and multiple-device configuration. You also can boot the PowerPC processor this way.

JTAG files are used with Xilinx download cables over a JTAG chain for the FPGA device(s) in your system. JTAG enables configuration, control and bitstream or debugging feedback. In particular, JTAG is the primary mechanism for the Xilinx ChipScope Pro 6.1i tool, which is an integrated logic analyzer that gets embedded into your on-chip logic for live in situ hardware debugging. You also can configure the programming of the FPGA directly from a PC, using the iMPACT tool and a download cable.

Linux and Interoperability

The tools also can be used on the command line, instead of with the GUI. Once the Xilinx binaries are on your Linux $PATH, you can drive them with batch mode scripts, programs and makefiles. If you want to run these commands from the shell, you can crib from the command log, produced by the Project

Navigator, and improvise from there. For example, these commands can be modified to add shell variables. For the design created in this article, the generated command log text looks like:

```
xst -intstyle ise -ifn __projnav/mpy16.xst \
-ofn mpy16.syr
ngdbuild -intstyle ise \
-dd /home/mab/VERILOG/MPY-TEST/_ngo -i  \
-p xc2vp7-fg456-6 mpy16.ngc mpy16.ngd
map -intstyle ise -p xc2vp7-fg456-6 -cm area \
-pr b -k 4 -c 100 -tx off \
-o mpy16_map.ncd mpy16.ngd mpy16.pcf
par -w -intstyle ise -ol std -t 1 mpy16_map.ncd \
mpy16.ncd mpy16.pcf
trce -intstyle ise -e 3 -l 3 -xml mpy16 mpy16.ncd \
-o mpy16.twr mpy16.pcf
bitgen -intstyle ise -f mpy16.ut mpy16.ncd
```

Conclusion

We have introduced FPGAs in the context of modern semiconductor technology, have described how software and hardware lines are blurring and have taken a quick tour of the ISE 6.1i FPGA implementation tools on Linux. You also can refer to the August 2002 issue of *LJ* (/article/6073), which first introduced the Xilinx ML300 development board and embedded Linux on the Virtex-II Pro FPGA.

Resources

ChipScope Pro: www.xilinx.com/ise/verification/chipscope_pro.htm

Design Tools: www.xilinx.com/ise/design_tools/index.htm

Development and Reference Boards: www.xilinx.com/xlnx/xebiz/board_search.jsp

Embedded Development Kit: www.xilinx.com/ise/embedded/edk.htm

Main Link: www.xilinx.com

Xilinx FPGA Devices: www.xilinx.com/xlnx/xil_prodcat_landingpage.jsp?title=Devices

Useful Verilog Resources

Icarus Verilog: icarus.com/eda/verilog

IEEE Std. 1364-1995 (ISBN 1-55937-727-5): standards.ieee.org

IEEE Std. 1364-2001 (ISBN 0-7981-280606): standards.ieee.org

Verilog Quickstart, Third Edition, by James M. Lee, Kluwer Academic Publishers, 2002. ISBN 0-7923-7672-2.

Verilog is a registered trademark of Cadence Design Systems, Inc.

Michael Baxter is technical editor of *Linux Journal*.

Archive Index Issue Table of Contents

Advanced search

RSTA-MEP and the Linux Crewstation

George Koharchik

Quintelle Griggs

Sonja Gross

Kathy Jones

John Mellby

Joe Osborne

Issue #114, October 2003

Automatically detect the enemy in the dark and notify friendly units where he is.

We recently completed a prototype Linux crewstation for the Reconnaissance, Surveillance and Target Acquisition Mission Equipment Package (RSTA-MEP). This article briefly describes the whole system, then focuses on the crewstation portion. The Raytheon RSTA-MEP program provides the capability to assess the battlefield quickly through real-time information from the fusion of onboard and offboard sensors. Advances in sensors and software provides for wide-area-search (WAS) imaging, automatic target detection (ATD) and aided target recognition (AiTR) capabilities. These capabilities provide the crew with real-time data, including target position, classification and priority. Combining this with the US Army's Tactical Internet allows the crew to formulate and contribute to a common operating picture of friendly and enemy forces. This vehicle is a technology demonstrator to show what emerging capabilities can be added into existing and future reconnaissance vehicles.

In its current incarnation, the vehicle has thermal sights to allow the user to see in the daytime or at night. The primary sensors on the mast are a long-range Forward Looking Infrared (FLIR) sensor, an Inertial Navigation System (INS) and a Global Positioning System (GPS) receiver. In addition to what's on the mast, there are several Raytheon NightSight infrared sensors attached to the vehicle so that the rest of the crew can look at the immediate area around the vehicle.

The mast is four meters high, and including the height of the vehicle, the sight is over five meters high. The vehicle has a three-member crew: driver, commander and scout/operator. The driver can also use the NightSight sensors to drive in the dark and look around for security purposes. The commander also has controls for the NightSight sensors, operates the connection to the Tactical Internet and directs the other two. The scout/operator uses the Linux crewstation to operate the mast-mounted sensors and their associated embedded systems.

The RSTA-MEP system is mounted onto an H1 Hummer and consists of a mast-mounted sight, embedded computers and a crewstation PC running Linux (Figure 1). These parts connect together as shown in Figure 2.

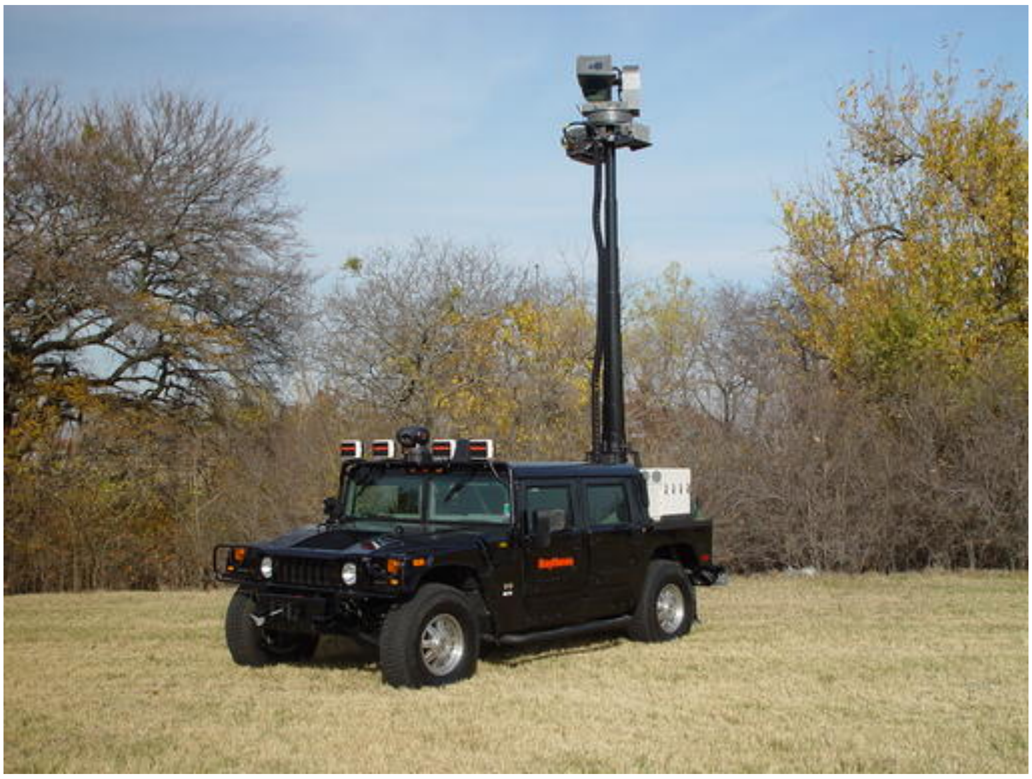


Figure 1. RSTA-MEP system mounted on a Hummer with mast extended. The sensors are at the top of the mast; the embedded systems are in white boxes on the back. The crewstation computer is inside the vehicle.

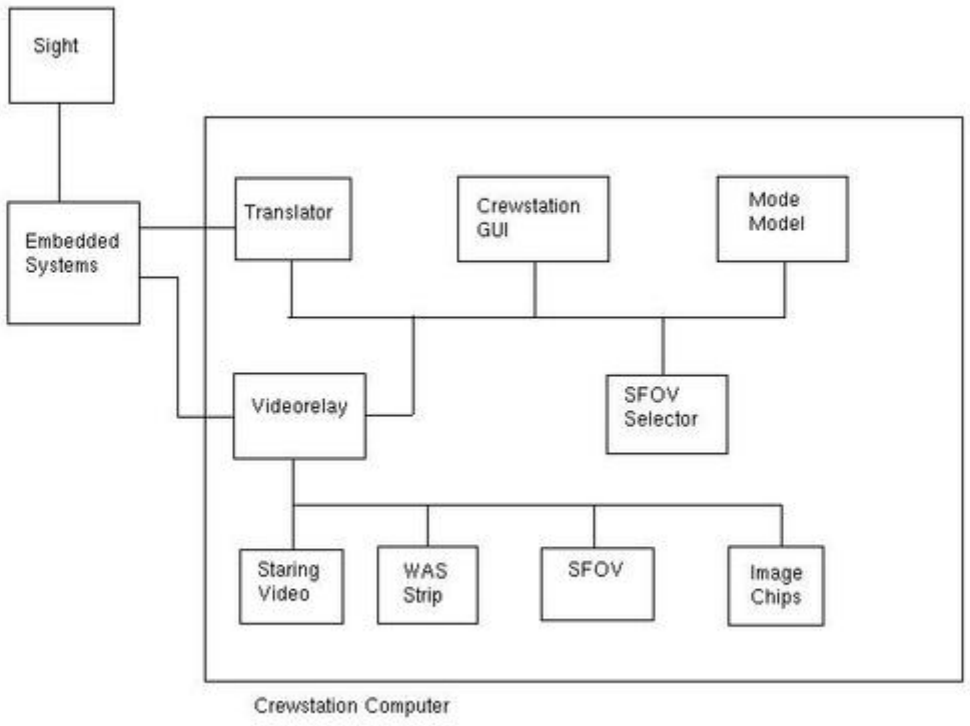


Figure 2. Crewstation Computer Connections and Modules

The Embedded Side

The embedded computers are digital signal processors that control the mechanics and electronics of the sensor (for example, pointing the sensor or cooling its detector) and some of the image processing. PowerPC boards running VxWorks and single-board computers running Microsoft Windows NT and Sun Solaris also are used. The applications running on these computers include the Force XXI Battle Command Brigade and Below (FBCB2, a US military digital command and control system), target detection and recognition, real-time image processing and communications. The package also includes a GPS receiver, inertial navigation system and digital map functionality. The embedded systems communicate with each other using Ethernet and Virtual Interface protocol (VI) on Fibre Channel.

Connecting to the Crewstation

The prototype Linux crewstation is the successor to an earlier system. The ideal case would have been for our new crewstation to fit in exactly the way the predecessor crewstation had. Our initial attempts to use VI on Fibre Channel failed. Our embedded systems group had considerable experience with vendor compatibility issues, so in selecting Fibre Channel hardware we were limited to vendors who supplied cards and drivers for both VxWorks and Linux. We couldn't find any of those who supported VI protocol. Our second attempt was to try to use disk emulation and imitate a hard drive connected to Fibre Channel, so we could at least stay on the same media.

The results there also were unsatisfactory, so we went to gigabit Ethernet. Ethernet would carry both the video from the sensor and the command and status data between the crewstation and the embedded systems. When looking at gigabit Ethernet, the home audience must consider four things: packet size, media, interconnecting and network interface card. Regular Ethernet has a maximum packet size of 1,500 bytes. An emerging standard for gigabit Ethernet is to allow a 9,000 byte maximum, called jumbo packets. For this project, our concerns about vendor compatibility between the embedded side and the Linux side pushed us to regular packet size.

The second consideration is media. Gigabit Ethernet cards come in two varieties, copper and optical fiber. Although copper is susceptible to electromagnetic interference (EMI), fiber is mechanically delicate. We chose copper because it's cheaper. If EMI became a problem, we always could get an optical card later without changing software. The choice of copper also meant we were compatible with our lab's infrastructure, courtesy of autonegotiation. Our existing network was 10/100 copper.

The third consideration is interconnect. The situation doesn't change too much from 10/100 Ethernet; you have switches, hubs and crossover cables. Switches route traffic so it's seen only by the intended recipient. They handle connections with differing speeds and duplexes, and they have the all-important blinky lights (the status lights on the switch that blink to show activity) to help with debugging. The disadvantages of switches are the cost and that you need a managed switch if you want to use a packet sniffer.

Hubs are the second choice. On the plus side, they are cheaper than switches and have the status lights. On the minus side, we know of no hubs for gigabit Ethernet (only switches), so if you use a 10/100 hub, you sacrifice speed. Hubs also send all packets everywhere, which is good if you're trying to sniff packets but bad if you're trying to limit the amount of traffic on an interface.

Crossover cables are the simplest option. They're the cheapest choice; they require no additional equipment, and you can be sure that no packets are coming from an outside source. On the other hand, there are no blinky lights, no way to connect an outside packet sniffer, and if one interface goes down (common for restarting embedded hardware), so does the other.

We chose switches, although the choice between switches and crossover cables is still a subject of religious debate. We also can pass on a caution about gigabit cabling. Professionally made category 5e or 6 cables are preferable to home-brew cables.

The fourth consideration is your network interface card; they generally come in 32- and 64-bit flavors. The 64-bit cards typically perform better with less draw on your PCI bus' resources. Although we didn't perform a trade study on available products, we chose the Intel Pro/1000 Server Adapter.

We chose to use TCP/IP on Ethernet. Although TCP is slower than UDP, it is a reliable protocol that compensates for any dropped, duplicated or reordered packets. We wanted to get the best-quality video in the face of possible EMI on the vehicle, so we deemed that built-in error correction was essential. Also, because no information is lost, duplicated or delivered out of order, command and status information would be reliable. When coding the socket layer for this, we had to tune the sizes of the socket send and receive buffers (using setsockopt with the SOL_SOCKET SO_RCVBUF and SO_SNDBUF options) to get enough throughput for the video. We also turned off Nagle's algorithm (setsockopt with IPPROTO_TCP and TCP_NODELAY) to reduce the latency between the crewstation and the embedded system, making it more responsive to sensor-pointing commands from the grips attached to the crewstation.

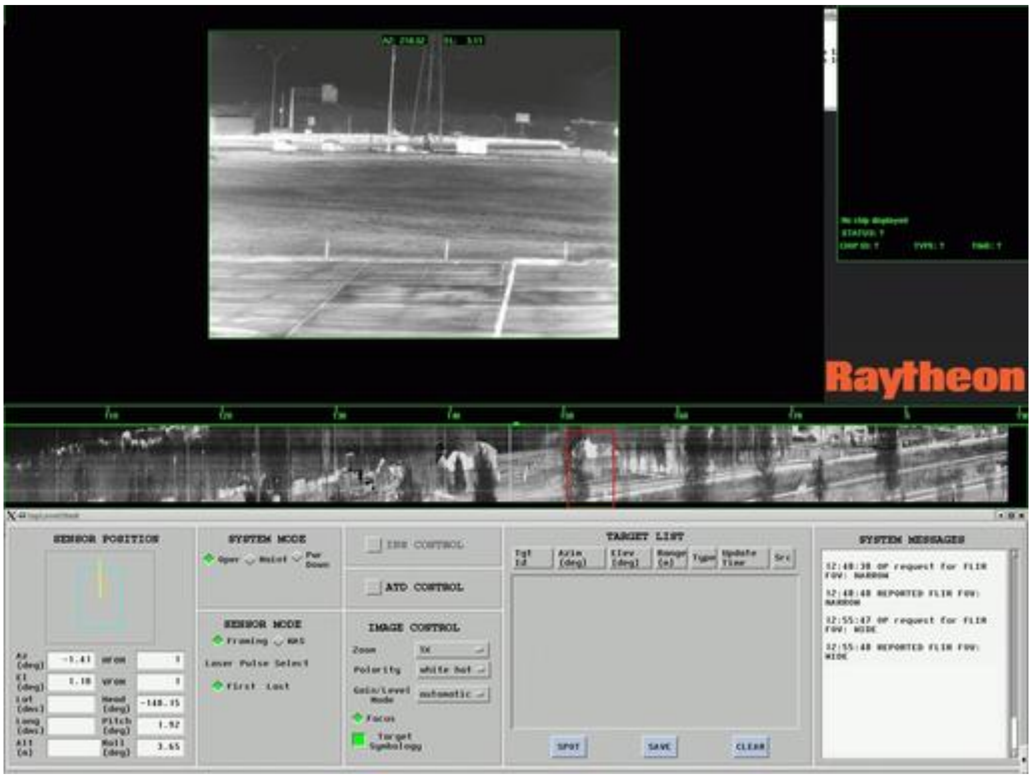
Crewstation Application Software

This prototype crewstation comes from Raytheon's Tiger simulator legacy, unlike the embedded side, which is Raytheon's implementation of the US Army's Weapons Systems Technical Architecture Working Group (WSTAWG) Common Operating Environment (COE). Although both the embedded side and the crewstation side are message-passing systems, the two messaging systems are not compatible. A translation module goes between the two. To minimize latency and CPU usage, this translator process is split into two threads using the POSIX threads library. One thread waits for input from the embedded side of a socket and translates it into the shared memory pool used by the crewstation modules. The other thread takes data from the shared memory pool and writes it to the socket for the embedded side to pick up. Dividing this work into two threads and using full compiler optimizations keeps the latency to a minimum.

The video relay module reads a separate gigabit Ethernet network connection devoted to video. It decides in which video window the data is to be displayed and routes it there.

The GUI control panel on the crewstation was generated by Builder's Xcessory (see Resources). Three major concerns drove the design of the GUI: limited screen space, reflecting the state of the embedded side and desire to use a grip instead of a mouse or trackball.

The first major design issue encountered was screen real estate. One monitor was used to display all imagery, leaving only the bottom third of the screen available for the GUI. The mode of the system and the controls for that mode are displayed in this third. The system has two major modes: WAS mode and conventional framing mode. In WAS mode the sensor quickly scans a user-selected area, and the grips allow the user to pick a section of that scanned area to be displayed as a super field of view (SFOV). In framing mode, live video is displayed, and the grips allow the user to point the sensor. As access to framing controls is not needed during WAS and vice versa, both sets of widgets were designed to occupy the same screen space. This is the sensor mode pane of the GUI in Figures 3 and 4. When one set is available for use, the other is hidden. Other functionality, such as controls for the automatic target detection software, was placed in separate windows and made accessible from buttons on the main GUI. These windows pop up over the image display area.



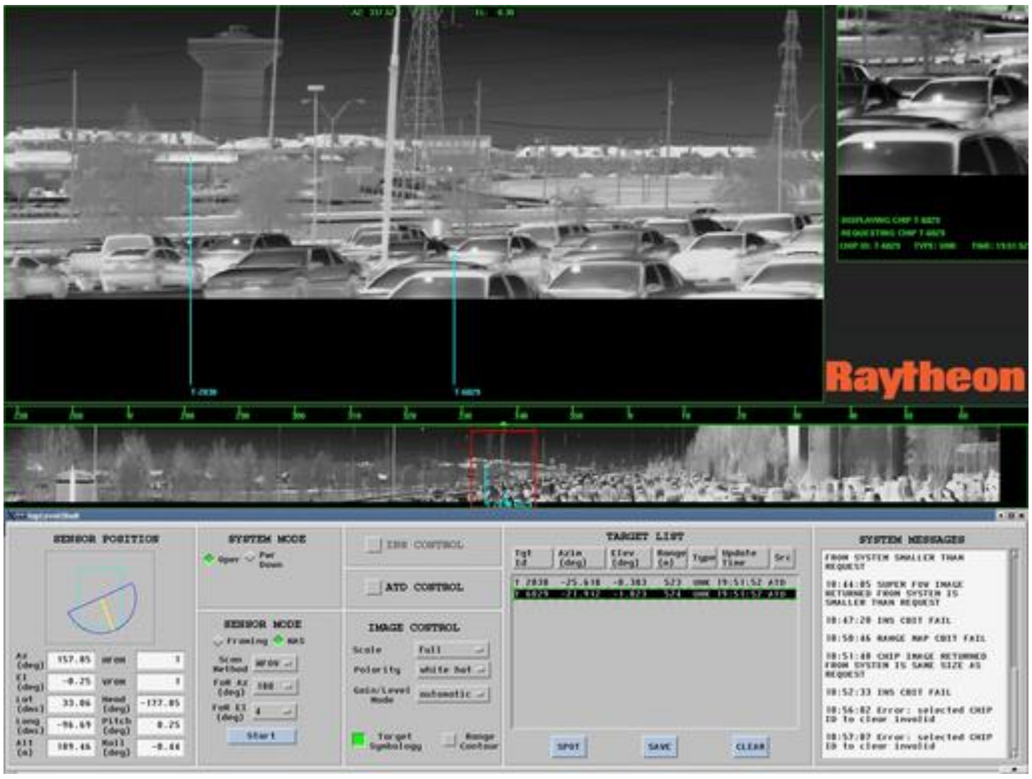
Az (deg)	-1.41	HFOV	1
El (deg)	1.18	VFOV	1
Lat (dms)		Head (deg)	-148.15
Long (dms)		Pitch (deg)	1.92
Alt (m)		Roll (deg)	3.65

Zoom	5X
Polarity	white hot
Gain/Level Mode	automatic

Tgt Id	Azim (deg)	Elev (deg)	Range (m)	Type	Update Time	Src

```
12:48:38 OP request for FLIR
FOV: NARROW
12:48:48 REPORTED FLIR FOV:
NARROW
12:55:47 OP request for FLIR
FOV: WIDE
12:55:48 REPORTED FLIR FOV:
WIDE
```

Figure 3. Crewstation with System in Framing Mode



Az (deg)	157.85	WFOM	1
El (deg)	-0.25	VFOM	1
Lat (deg)	33.06	Head (deg)	-177.85
Long (deg)	-96.69	Pitch (deg)	0.25
Alt (m)	189.46	Roll (deg)	-0.46

Scan Method: WFOV
FoR Az (deg): 180
FoR El (deg): 4

Scale: full
Polarity: white hot
Gain/Level Mode: automatic

Tgt Id	Azim (deg)	Elev (deg)	Range (m)	Type	Update Time	Src
T 2838	-25.618	-0.383	523	UNK	19:51:52	ATD
T 6829	-21.912	-1.823	524	UNK	19:51:57	ATD

```
FROM SYSTEM SMALLER THAN
REQUEST

18:44:05 SUPER FOV IMAGE
RETURNED FROM SYSTEM IS
SMALLER THAN REQUEST

18:47:20 INS CBIT FAIL

18:50:46 RANGE MAP CBIT FAIL

18:51:48 CHIP IMAGE RETURNED
FROM SYSTEM IS SAME SIZE AS
REQUEST

18:52:33 INS CBIT FAIL

18:56:02 Error: selected CHIP
ID to clear invalid

18:57:07 Error: selected CHIP
ID to clear invalid
```

Figure 4. Crewstation with System in WAS Mode

This lack of screen space also presents a second problem. There is a need for immediate visual cues of system response to user input as well as a report of the current state of the embedded side. Rather than use separate widgets for control and status objects, the same widget is used for both. When the operator manipulates a widget, the operator's command is reflected automatically on the GUI, and the widget's callback code is triggered. The callback sends a message containing the requested change to the mode model. This request is passed to the embedded sensor side, which then returns a status. Should the status differ from the request, the mode model notifies the GUI, which in turn updates the widget to display the correct status value.

A third design issue is the need for a mouseless environment. Vehicle movement and lack of physical desktop space make it difficult to use a mouse, trackball or touchscreen. A keyboard is available but is used only for minimal data entry. For these reasons, we desire to manipulate the GUI with the hand grip.

Mouseless mode was accomplished in an early version of the GUI by adding manual widget traversals and button-press events. Moving the hat switch on the grip changed widget focus via XmProcessTraversal calls. Pressing the Select button on the grip defined and sent an XEvent, similar to this:

```
/* sending key press events */
#include <X11/keysym.h>

XKeyEvent ev;
```

```
   Window    rootWin;
   int       x,y;
   int       root_x,root_y;
   Window    win;

   rootWin =  RootWindowOfScreen (guiScreen);
   win = findPointerWindow(rootWin, &x, &y,
                           &root_x, &root_y);

   ev.type = (long) KeyPress;
   ev.send_event = True;
   ev.display = display;
   ev.window = win;
   ev.root = rootWin;
   ev.subwindow = 0;
   ev.time = CurrentTime;
   ev.x = 1;
   ev.y = 1;
   ev.x_root = 1;
   ev.y_root = 1;
   ev.state = 0;
   ev.same_screen = True;

   ev.keycode = XKeysymToKeycode(display,XK_space);
   XSendEvent(display, window, True, KeyPressMask,
              (XEvent *)&ev);
```

Unlike the current version of the GUI, the previous version consisted of a single topLevelShell that contained only simple widgets, for example, PushButtons and ToggleButtons. The current GUI includes multiple shells (pop-up windows) and composite widgets, such as OptionMenus. Simply calling XmProcessTraversal to change focus does not work across shells. Sending a button press on an OptionMenu pops up the menu choices; however, sending a second button press does not select the option and does not pop down the menu.

The home audience should keep these facts in mind:

1. The window manager is the boss. When dealing with multiple shells, remember that window managers do not readily relinquish control of the focus—or anything else for that matter.
2. Widget hierarchy has an effect. The order of traversal in a group of widgets is determined partially by the order in which they were declared in your (or BX's) code.
3. Be aware of behind-the-scenes code. Consider a RadioBox containing two ToggleButton children, with toggle A selected. When an incoming message to select toggle B is received, simply swapping the values of the children's XmNset resources looks correct on screen. The parent widget, however, still thinks toggle A is selected, which can lead to unexpected Button behavior.

In the current version of our project, a separate process takes input from the hand grip and controls the mouse pointer using a combination of XWarpPointer and the X server's XTest extensions (see Resources). The crewstation also displays video generated from data sent by the embedded side. The video relay process reads this from a socket and dispatches it to a window. There are four windows: framing video, WAS, SFOV and image chips.

As mentioned above, the framing video is live image data. The image in the WAS window is a compressed image strip that represents an image taken by a rapid scan of the sensor across a fixed area. Symbology in the WAS strip includes locations where the targeting systems think there might be targets. The SFOV is a larger view of a user-selected section of the WAS strip that shows more detail. Target symbology and information from the digital map are visible here. Image chips are segments of the scene where the targeting system finds something interesting. These are presented to the operator for evaluation and reporting to other systems that are off-vehicle. Figure 3 shows an outdoor view with the system in framing mode with the GUI, framing video and WAS strip. Figure 4 shows the system in WAS mode with the WAS strip, SFOV, GUI and an image chip window.

Video is implemented in OpenGL as a texture on a polygon, and the data from the video is put into an OpenGL texture and applied to the polygon. Then, when the polygon is drawn you see the image. (See the example code in Listing 1, available at ftp.linuxjournal.com/pub/lj/listings/issue114/6634.tgz, to illustrate the technique.) We chose OpenGL for video because it offers us a lot of options for processing and displaying the data. The image can be resized or rotated if the data is generated on a different orientation from how it's displayed. OpenGL has a lot of primitives for drawing symbology on top of the image, some image-processing ability built-in and double buffering for flicker-free updates. OpenGL is portable and well documented. Additionally, we can off-load a lot of the work from the CPU onto the graphics card hardware.

The SFOV selector controls what part of the WAS strip picture is selected for display on the SFOV window. It also controls where the red rectangle is drawn in the WAS strip window.

The crewstation has a separate control and moding module. Instead of having pieces of logic scattered throughout the modules in the system, they are concentrated in this one module. This design makes the other parts of the system simpler and more reusable. It also makes the moding module fiendishly complex. The mode model has to embody the knowledge of how the other pieces interact and mirror the state of the embedded system and the crewstation. It allows the crewstation to make permissible actions based on that state and monitors the embedded side for errors and unexpected state changes.

How Fast Is Fast Enough?

The crewstation falls into the category of soft real time: the system doesn't fail if something is late. Because this is a human-in-the-loop test bed, with most of the time-critical components in the embedded systems, it has to run only fast enough for the operator to perform tasks. For this reason we're not using one of the real-time Linux frameworks; we're overpowering the problem with brute-force hardware. We have SCSI disks, enough memory basically to eliminate paging, a GeForce4 graphics card for fast OpenGL and dual 2.4GHz processors from Microway.

A limit did emerge for two parts of the system: video and pointing the sensor. The live framing video is fed to the crewstation at RS-170 rate, one set of scan lines every 1/60 of a second. These had to be combined and displayed fast enough to keep up and maintain a constant rate. To do that, we made sure we had enough network bandwidth to ship the video and plenty of CPU and graphics horsepower to keep the displays refreshed. We then used the NVIDIA driver's ability to sync to the vertical retrace of the monitor. With the monitor set to refresh at 60Hz, we were there. (See the README.txt file supplied with NVIDIA drivers; the current one is at download.nvidia.com/XFree86_40/1.0-4194/README.)

Pointing the sensor presents a similar challenge. The sensor must be responsive enough to the grips that the operator can point it without missing the target and overcompensating. Although the video is largely a matter of bandwidth, pointing the sensor is a matter of latency. Having a long message chain contributes to latency. For instance, a button press or a slewing command starts off in the grip or GUI process, goes to the control process to determine if that input is valid and then moves to the translator to go to EO message format. Next it goes across the gigabit Ethernet to an embedded process that receives that message, then moves to the OE and the embedded systems code, then on to the actuator and, finally, the results come back in the video stream. The combination of dividing the translator process into two threads and compiling with full optimizations (`-Wall -ansi -O3 -ffast-math -mpentiumpro`) did the trick.

We used the gprof profiler to see where the hot spots were in the code. (See the info page for gprof.) Here, we ran into a problem with profiling the video code: when we used X timers (XtAppAddTimeOut), no timing data accrued in the profile. (Do the profiler and XtAppAddTimeOut use the same signal and interfere with each other?) Another optimization we discovered is for the video source to write both the odd and even scan lines across the network with a single write statement instead of two separate, smaller ones.

Pluses, Pitfalls and Conclusions

Using Linux led to problems in a few places. For instance, we couldn't find a vendor who supplied PCI Mezzanine cards for PowerPC with VxWorks drivers, PCI cards with Linux drivers or who could handle VI protocol. In the end we had to drop Fibre Channel.

We did find, though, a couple of cases where Linux gave us an advantage on this project. Because we booted from a hard drive, we didn't have to write our system to EEPROM the way the embedded side did. When they made that transition to EEPROM, their ability to debug was diminished. Also, Linux provides core files to aid debugging, which VxWorks doesn't. The Linux crewstation is more robust and delivers better image quality than its predecessor. Finally, our shop integration and unit testing are easier on Linux, because commodity PCs are more plentiful than are embedded PowerPCs. In the future, we expect that having the full power and flexibility of the Linux, X and OpenGL environment will be valuable as we add more modes and more devices to our prototype.

Resources

BX is a trademark of Integrated Computer Solutions Inc.: www.ics.com and www.ics.com/products/bxpro

GeForce is a trademark of the NVIDIA Corporation: www.nvidia.com

Microway is a trademark of Microway Inc.: www.microway.com

Power Programming...Motif by Eric F. Johnson and Kevin Reichard, Second Edition Version 1.2, 1993, Management Information Source, Inc. New York, New York. ISBN: 1-55828-319-6.

Raytheon and NightSight are trademarks of the Raytheon Company: www.raytheon.com and www.raytheon.com/products/tiger

Unofficial VxWorks and Tornado FAQ: www.xs4all.nl/~borkhuis/vxworks/vxfaq.html

VxWorks and Tornado are trademarks of Wind River systems: www.windriver.com

VxWorks/Tornado II FAQ (especially section 4.6 on sockets): www.xs4all.nl/~borkhuis/vxworks/vxworks.html

WSTAWG Web Page: wstawg.army.mil/index.asp

XTest Extensions to XFree86: xfree86.org/pub/XFree86/4.2.0/doc/xtestlib.TXT

George Koharchik (g-koharchik@raytheon.com) works for Raytheon's Visualization and Simulation Lab (VSL) and spends his spare time mulling over the mechanics of safety pins.

Quintelle Griggs (Quintelle_Y_Griggs@raytheon.com) works at Raytheon's VSL.

Sonja Gross (sonja_gross@raytheon.com) joined Raytheon's VSL in 2001 after earning her Bachelor's degree in Computer Science from Louisiana Tech University.

Kathy Jones (kajones@raytheon.com) is a software developer at Raytheon's VSL. She develops Motif and VAPS UIs and other software tools.

John Mellby (j-mellby@raytheon.com) works at Raytheon's VSL on virtual simulation architectures, writing short biographies and measuring the Texas sun during the spring equinox.

Joe Osborne (joe.osborne@smiths-aerospace.com) works at Smiths Aerospace in Grand Rapids, Michigan.

Archive Index Issue Table of Contents

Advanced search

Revisiting Old APIs

Greg Kroah-Hartman

Issue #114, October 2003

Some of the kernel APIs we covered in their 2.5 form have changed a little before being nailed down for 2.6. Here's the updated information on the 2.6 versions.

It has been over a year since this column started, and due to the rapid rate of Linux kernel development, much of what was written back then is now wrong. This month, we cover the changes in the different kernel APIs that have been discussed previously.

tty Changes

The tty layer had been one of the most stable kernel APIs around, and it showed. The lack of proper reference counting, the locking shoved in where it looked needed and the strange way tty devices were allocated are all due to its age. Thankfully, Al Viro recently cleaned up a lot of the old cruft in the tty layer in the 2.5 kernel series. Because of this, a number of things have changed for anyone wanting to write a new tty driver.

In the August and October 2002 issues of *LJ* [available at /article/5896 and /article/6226], we discussed the tty layer and how to fill up the struct tty_driver structure with all the necessary function callbacks. Since then, a new structure, struct tty_operations, has been created to hold all of the function callbacks. The struct tty_driver still contains the older function pointers, so a new function has been created to copy these pointers over, tty_set_operations. Hopefully, this duplication will be eliminated soon.

A number of variables have been removed from the struct tty_driver. The table, termios, termios_locked and refcount fields are gone. The tty layer now handles all of the proper locking and reference counting, without forcing the individual tty drivers to allocate the space for these locks.

The magic and num variables no longer need to be set explicitly by the tty driver. These variables now are set in a new function, alloc_tty_driver, that must be called by all tty drivers to allocate the space for the tty driver. The number of different tty devices this driver is going to support is passed as a parameter to the function. For example, the tiny tty driver introduced in the original tty articles would create the struct tty_driver as:

```
/* allocate the tty driver */
tiny_tty_driver = alloc_tty_driver(TINY_TTY_MINORS);
```

In the past, picking the proper name for the tty driver was difficult, as the devfs kernel option overloaded the use of the name field. Christoph Hellwig finally fixed this mess by introducing a new variable in the struct tty_driver structure called devfs_name. Now, the name field should be set to a simple, small name that shows up in the tty proc files. The devfs_name should be set to the name that devfs uses to generate the device node for this driver.

In the 2.5 kernel series, the MOD_INC_USE_COUNT and MOD_DEC_USE_COUNT macros were declared too racy, and almost all usage of them in the kernel was removed. In order to do this, module reference counting was pushed a layer higher than the original calls. This allows the kernel to increment a module's reference count before the kernel jumps into the module. Likewise, when the kernel is finished with the module, it knows to decrement the count automatically.

This module change was done in the tty layer, so no tty driver should contain the MOD_* macros. Instead, an owner variable was added to the struct tty_driver to show what module owns the tty driver. The following line shows how to set this variable properly:

```
tiny_tty_driver->owner = THIS_MODULE;
```

This tells the tty core what module is related to this tty driver.

As for these tty changes, here is how to initialize and register a tty driver with the kernel properly:

```
#define TINY_TTY_MAJOR240 /* experimental range */
#define TINY_TTY_MINORS 255
                        /* use the whole major up */

static struct tty_operations serial_ops = {
    .open =       tiny_open,
    .close =      tiny_close,
    .write =      tiny_write,
    .write_room = tiny_write_room,
};

static struct tty_driver *tiny_tty_driver;

static int __init tiny_init(void)
{
    /* allocate the tty driver */
    tiny_tty_driver =
        alloc_tty_driver(TINY_TTY_MINORS);

    /* initialize the tty driver */
    tiny_tty_driver->owner = THIS_MODULE;
    tiny_tty_driver->driver_name = "tiny_tty";
    tiny_tty_driver->name = "ttty";
    tiny_tty_driver->devfs_name = "tts/ttty%d";
    tiny_tty_driver->major = TINY_TTY_MAJOR,
    tiny_tty_driver->type = TTY_DRIVER_TYPE_SERIAL,
    tiny_tty_driver->subtype = SERIAL_TYPE_NORMAL,
    tiny_tty_driver->flags =
        TTY_DRIVER_REAL_RAW | TTY_DRIVER_NO_DEVFS,
    tiny_tty_driver->init_termios = tty_std_termios;
    tiny_tty_driver->init_termios.c_cflag =
        B9600 | CS8 | CREAD | HUPCL | CLOCAL;
    tty_set_operations(tiny_tty_driver, &serial_ops);
    if (tty_register_driver(tiny_tty_driver)) {
       printk(KERN_ERR
              "failed to register tiny tty driver");
       return -1;
    }

    printk(KERN_INFO DRIVER_DESC " " DRIVER_VERSION);
    return 0;
}

static void __exit tiny_exit (void)
{
    tty_unregister_driver(tiny_tty_driver);
}

module_init(tiny_init);
module_exit(tiny_exit);
```

tty callout devices also are now removed entirely from the kernel. Any tty driver living in the 2.5 kernel tree that used callout support has had it removed.

Fewer ioctls

Along with the tty structure changes, a few tty ioctls have been removed, specifically the TIOCMGET, TIOCMBIS, TIOCMBIC and TIOCMSET ioctls. They have been replaced with two new function callbacks, tiocmget and tiocmset, that have been added to the struct tty_operations structure. These functions are defined as:

```
int (*tiocmget)(struct tty_struct *tty,
                struct file *file);
int (*tiocmset)(struct tty_struct *tty,
                struct file *file,
                unsigned int set,
                unsigned int clear);
```

The tiocmget function is called when the tty core or a user wants to know what the current line settings are for a specific tty port. This works almost exactly like the old TIOCMGET ioctl call did. The line status is defined as the different MSR_* values, but instead of being copied back into user space, as the old ioctl required the driver to do, they merely are returned from the function call. Here's an example of how a tiocmget function could look:

```
int
tiny_tiocmget(struct tty_struct *tty,
              struct file *file)
{
    struct tiny_private *tp = tty->private;
    unsigned int msr = tp->msr;
    unsigned int mcr = tp->mcr;
    unsigned int result = 0;

    result = ((mcr & MCR_DTR)    ? TIOCM_DTR: 0)
                                /* DTR is set */
              | ((mcr & MCR_RTS) ? TIOCM_RTS: 0)
                                /* RTS is set */
              | ((msr & MSR_CTS) ? TIOCM_CTS: 0)
                                /* CTS is set */
              | ((msr & MSR_CD)  ? TIOCM_CAR: 0)
                     /* Carrier detect is set */
              | ((msr & MSR_RI)  ? TIOCM_RI:  0)
                     /* Ring Indicator is set */
              | ((msr & MSR_DSR) ? TIOCM_DSR: 0);
                                /* DSR is set */

    return result;
```

The tiocmset function is called when the tty core or a user wants to set or clear any of the different line settings. This single function replaces the TIOCMBIS, TIOCMBIC and TIOCMSET ioctl calls. The set and clear variables in this function are used to tell which line settings to set and which to clear. The same line setting cannot be asked to be cleared and set at the same time, so the order in which variables are processed does not matter. An example of the tiocmset function is:

```
int
tiny_tiocmset(struct tty_struct *tty,
              struct file *file,
              unsigned int set, unsigned int clear)
{
    struct tiny_private *tp = tty->private;

    if (set & TIOCM_RTS)
        mcr |= MCR_RTS;
    if (set & TIOCM_DTR)
        mcr |= MCR_RTS;
    if (set & TIOCM_LOOP)
        mcr |= MCR_LOOPBACK;

    if (clear & TIOCM_RTS)
        mcr &= ~MCR_RTS;
    if (clear & TIOCM_DTR)
        mcr &= ~MCR_RTS;
    if (clear & TIOCM_LOOP)
        mcr &= ~MCR_LOOPBACK;

    /* set the new MCR value in the device */
    tp->mcr = mcr;
    return 0;
}
```

The usbserial core also has been affected a bit by these tty core changes. The tiocmget and tiocmset functions have been added to the struct usb_serial_device_type structure. The tty calls to these functions are passed down to the lower usbserial drivers, if the usbserial driver provides those callbacks.

Acknowledgements

I would like to thank Al Viro, Christoph Hellwig and Russell King for finally starting to work on cleaning up the tty layer to raise it to the proper standards of the rest of the kernel. Their changes have been instrumental in simplifying the tty driver interface, allowing driver authors to focus on the specific hardware implementations and not worry about the kernel interactions as much as before.

Greg Kroah-Hartman currently is the Linux kernel maintainer for a variety of different driver subsystems. He works for IBM, doing Linux kernel-related things, and can be reached at greg@kroah.com.

Kernel Korner

Using RCU in the Linux 2.5 Kernel

Paul E. McKenney

Issue #114, October 2003

Read-copy update, a synchronization technique optimized for read-mostly data structures, is new with the 2.5/2.6 kernel and promises better SMP scalability.

The Linux hacker's toolbox already contains numerous symmetric multiprocessing (SMP) tools, so why bother with read-copy update (RCU)? Figure 1 answers this question, presenting hash-lookup performance with per-bucket locks on a four-CPU, 700MHz Pentium III system. Your mileage will vary with different workloads and on different hardware. For an excellent write-up on the use of other SMP techniques, see Robert Love's article in the August 2002 issue of *Linux Journal* [available at /article/5833].

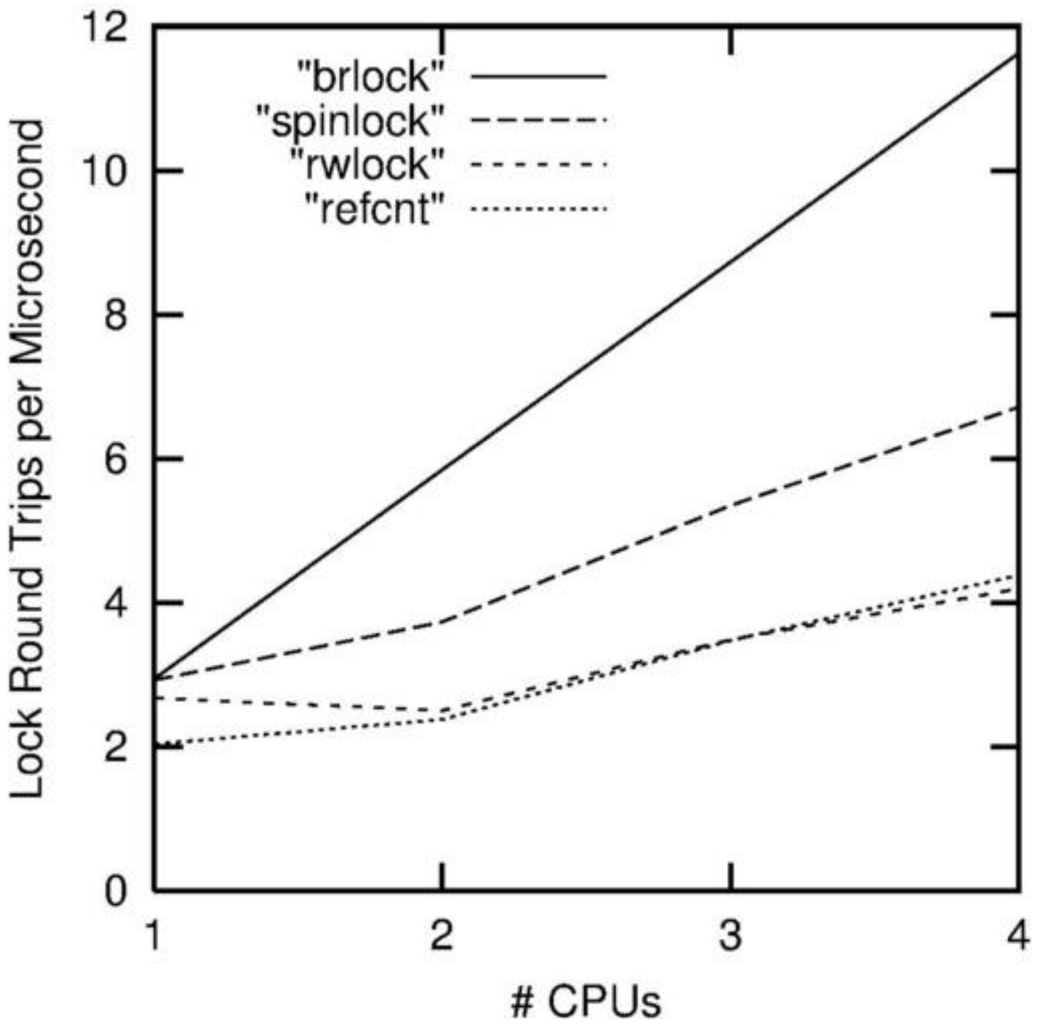


Figure 1. Hash-lookup performance scales poorly with number of CPUs.

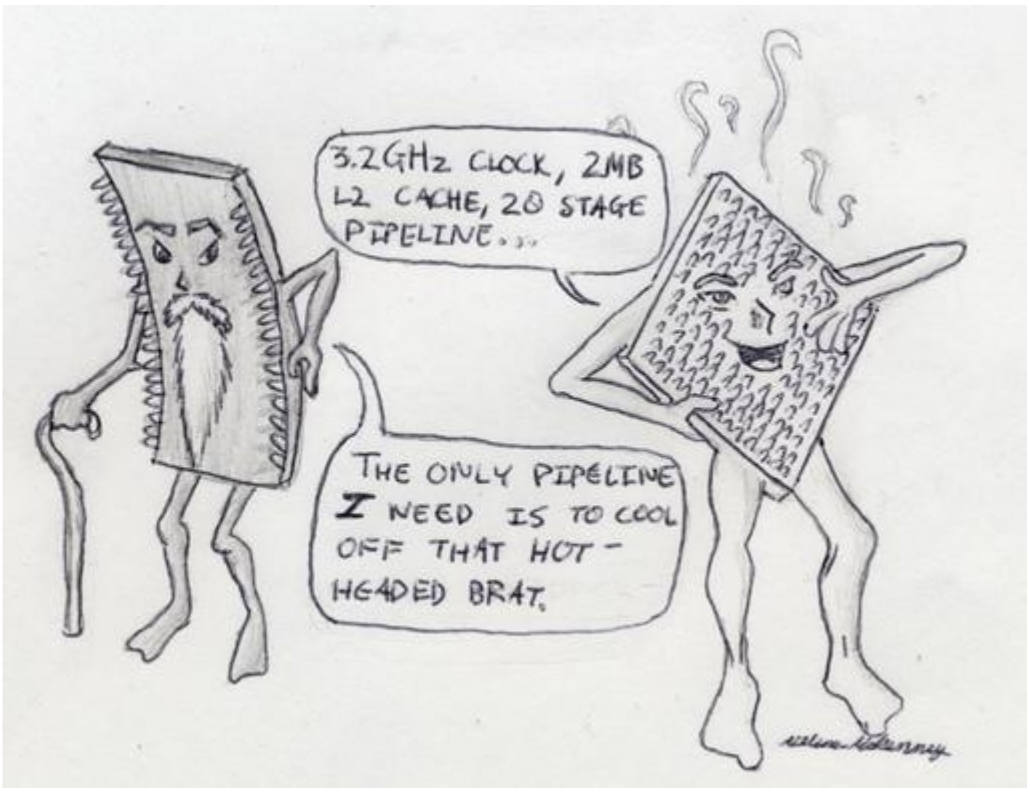
All accesses are read-only, so one might expect rwlock to work as well as this system. However, one would be mistaken; rwlock actually scales negatively from one to two CPUs, partly because this variant of rwlock avoids starvation, thus incurring greater overhead. A much larger critical section is required for rwlock to be helpful. Although rwlock beats refcnt (a spinlock and reference counter) for small numbers of CPUs, even refcnt beats rwlock at four CPUs. In both cases, the scaling is atrocious; refcnt at four CPUs achieves only 54% of the ideal four-CPU performance, and rwlock achieves only 39%.

Simple spinlock incurs less overhead than either rwlock or refcnt, and it also scales somewhat better at 57%. But this scaling is still quite poor. Although some spinning occurs, due to CPUs attempting to access the same hash chain, such spinning accounts for less than one-quarter of the 43% degradation at four CPUs.

Only brlock scales linearly. However, brlock's single-CPU performance is subpar, requiring more than 300 nanoseconds to search a single-element hash chain with simple integer comparison. This process should not take much more than 100ns to complete.

Not Your Parents' Microprocessor

Figure 2 illustrates the past quarter century's progress in hardware performance. The features that make the new kids (brats) so proud, however, are double-edged swords in SMP systems.


3.2GHz CLOCK, 2MB L2 CACHE, 20 STAGE PIPELINE...
THE ONLY PIPELINE I NEED IS TO COOL OFF THAT HOT-HEADED BRAT.
Figure 2. New, fast “brat” processors change the OS design rules.

Unfortunately, many algorithms fail to take advantage of the brat's strengths, because they were developed back when the old man was in his prime. Unless you like slow, stately computing, you need to work with the brat.

The increase in CPU clock frequency has been astounding—where the old man might have been able to interfere with AM radio signals, the young brat might be able to synthesize them digitally. But memory speeds have not increased nearly as fast as CPU clock rates, so a single DRAM access can cost the brat up to a thousand instructions. Although the brat compensates for DRAM latency with large caches, these caches cannot help data bounced among CPUs. For example, when a given CPU acquires a lock, the lock has a 75% chance of being in another CPU's cache. The acquiring CPU stalls until the lock reaches its cache.

Cacheline bouncing explains much of the scaling shortfall in Figure 1, but it does not explain poor single-CPU performance. When there is only one CPU, no other caches are present in which the locks might hide. This is where the brat's 20-stage pipeline shows its dark side. SMP code must ensure that no critical section's instructions or memory operations bleed out into surrounding code.

After all, the whole point of a lock is to prevent multiple CPUs from concurrently executing any of the critical section's operations.

Memory barriers prevent such bleeding. These memory barriers are included implicitly in atomic instructions on x86 CPUs, but they are separate instructions on most other CPUs. In either case, locking primitives must include memory barriers. But these barriers cause pipeline flushes and stalls, the overhead of which increases with pipeline length. This overhead is responsible for the single-CPU slowness shown in Figure 1.

Table 1 outlines the costs of basic operations on 700MHz Intel Pentium III machines, which can retire two integer instructions per clock. The atomic operation timings assume the data already resides in the CPU's cache. All of these timings can vary, depending on the cache state, bus loading and the exact sequence of operations.

Table 1. Time Required for Common Operations on a 700MHz Pentium III

Operation	Cost (ns)
Instruction	0.7
Clock cycle	1.4
L2 cache hit	12.9
Atomic increment	58.2
cmpxchg atomic increment	107.3
Main memory	162.4
CPU-local lock	163.7
Cache transfer	170.4–360.9

The overheads increase relative to instruction execution overhead. For example, on a 1.8GHz Pentium 4, atomic increment costs about 75ns—slower than the 700MHz Pentium III, despite having a more than twice as fast clock.

These overheads also explain rwlock's poor performance. The read-side critical section must contain hundreds of instructions for it to continue executing once some other CPU read acquires the lock, as illustrated in Figure 3. In this figure, the vertical arrows represent time passing on two pairs of CPUs, one pair using rwlock and the other using spinlock. The diagonal arrows represent data moving between the CPUs' caches. The rwlock critical sections do not overlap at all; the overhead of moving the lock from one CPU to the other rivals that of the critical section.

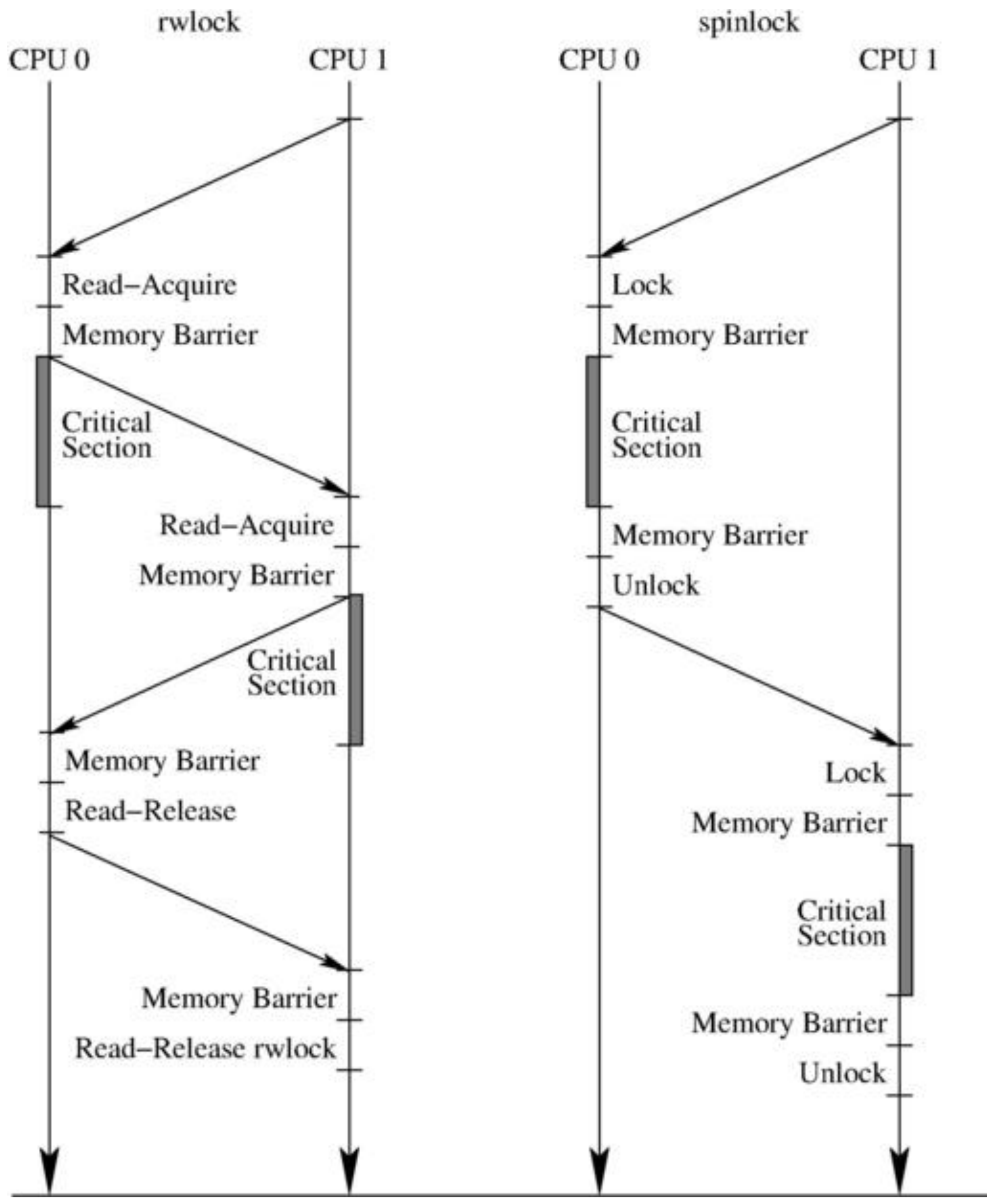


Figure 3. Timelines for rwlock and Spinlock on Two-CPU Systems

Lesson: Avoid Expensive Operations

If you care about performance, you want to avoid these expensive operations. Avoiding them is precisely what RCU does, at least for read-only accesses to read-mostly data structures, although the DEC Alpha still requires some read-side memory barriers. As seen in Figure 4, RCU scales well and has good single-CPU performance for the hash-table-search benchmarklet.

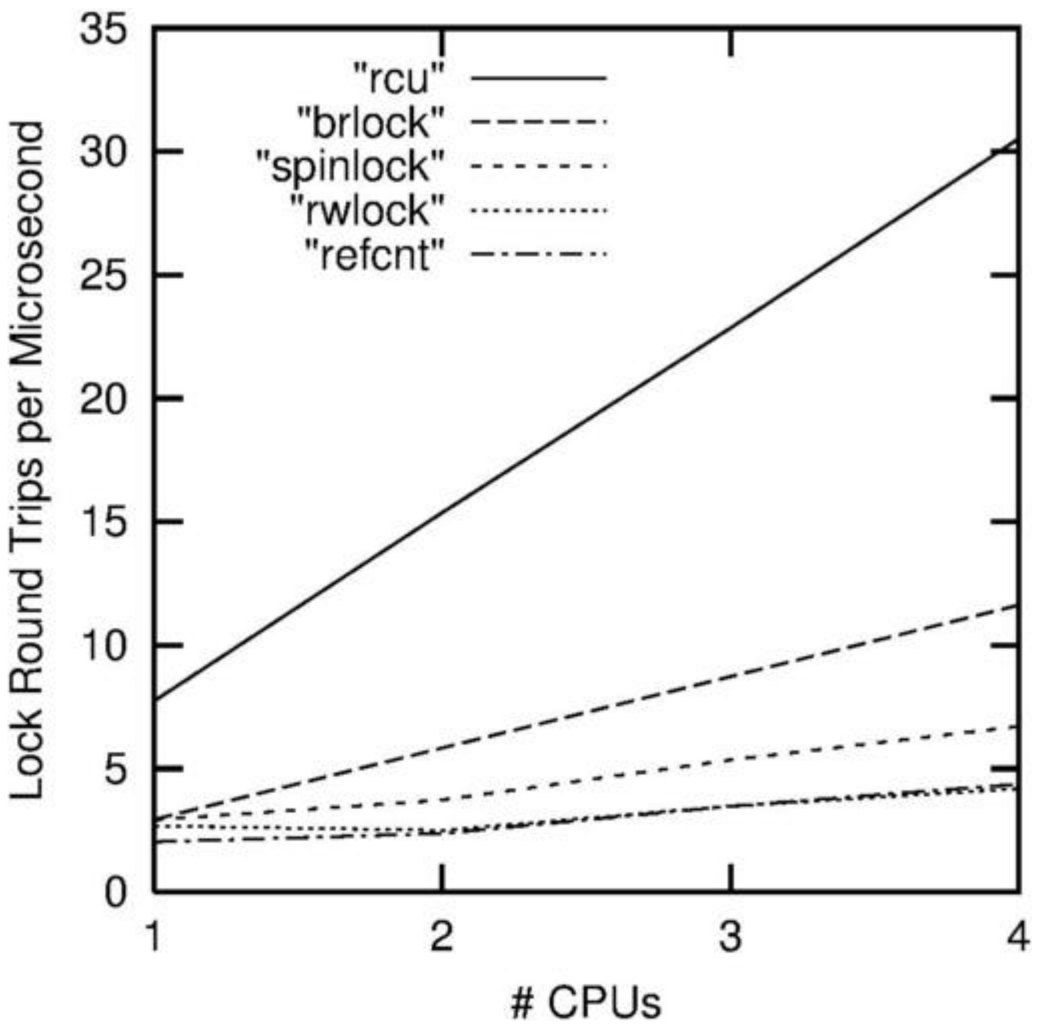


Figure 4. RCU Read Performance by Number of Processors

Of course, updates do slow down RCU, as shown in Figure 5. This graph illustrates the relative performance of these synchronization primitives as the workload varies from read-only (left-hand side) to write-only (right-hand side). RCU is better than brlock across the board. In fact, RCU has replaced brlock in the 2.5 kernel, thanks to Steve Hemminger of OSDL and a number of Linux's networking luminaries. RCU is the best option overall as long as fewer than about one-third of the accesses are updates. Again, your mileage will vary depending on your workload and hardware. In particular, workloads with greater per-element local processing—for example, more complex comparisons—would scale better. As always, use the right tool for the job.

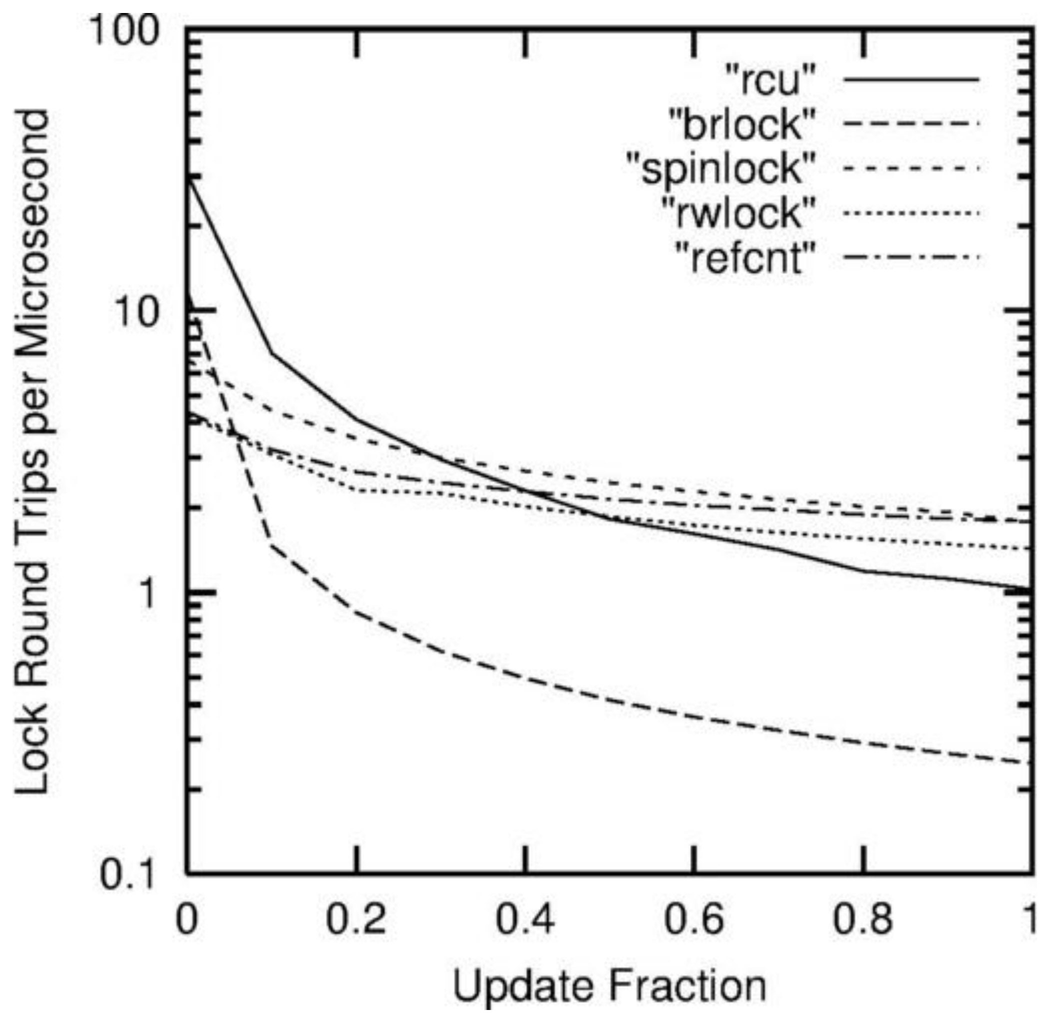


Figure 5. RCU Performance by Fraction of Accesses That Are Updates

How Does RCU Work?

If reading CPUs never make their presence known, how can updating CPUs avoid messing up readers? With locks, the updating CPU examines the lock state to determine when it is safe to carry out the update. With RCU, the updating CPU must make this determination indirectly.

The trick is RCUs reading CPUs are not permitted to block while traversing the data structure, the same as when CPUs holding a spinlock or rwlock are not permitted to block. This means that once an element is unlinked from a list, any CPU that subsequently performs a context switch cannot possibly gain a reference to this element. Context switch is a quiescent state: CPUs undergoing context switches cannot hold references to RCU-protected data structures. Any time period during which all CPUs pass through a quiescent state is a grace period. A CPU may therefore free up an element after a grace period has elapsed from the time that it unlinked the element from the list.

Thus, a simple, though inefficient, RCU-based deletion algorithm could perform the following steps in a non-preemptive Linux kernel:

- Unlink element B from the list, but do not free it—the state of the list as shown in Step 2 of Figure 6.
- Run on each CPU in turn. At this point, each CPU has performed one context switch after element B has been unlinked. Thus, there cannot be any more references to element B, as shown in Step 3 (Figure 6).
- Free up element B, as shown in Step 4 (Figure 6).

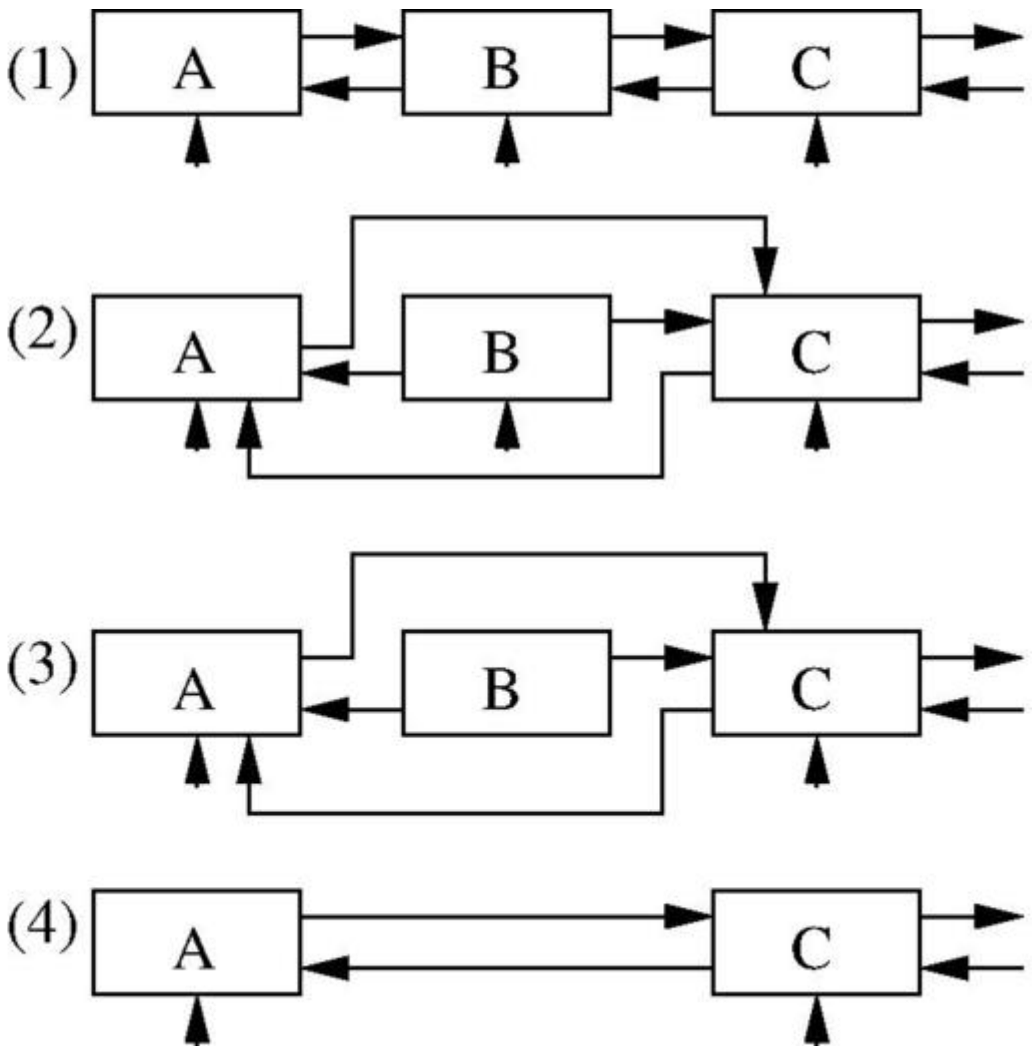


Figure 6. Steps of a Simple RCU-Based Deletion Algorithm

Andrea Arcangeli created a more efficient algorithm that boasts extremely short grace periods, which was the first Linux RCU implementation shipped. Dipankar Sarma coded up an even more efficient RCU implementation that maintains callback cache locality and permits a grace period to service any number of concurrent updates. Dipankar's algorithm is included in the 2.5 kernel and was described in detail at the Ottawa Linux Symposium in 2002.

RCU API

Listing 1 shows the external API for RCU. The synchronize_kernel() function blocks for a full grace period. This is an easy-to-use function, but it incurs expensive context-switch overhead. It also cannot be called with locks held or from interrupt context. However, it does allow concurrent callers to share a grace period.

Listing 1. The RCU API

```
void synchronize_kernel(void);

void call_rcu(struct rcu_head *head,
              void (*func)(void *arg),
              void *arg);

struct rcu_head {
        struct list_head list;
        void (*func)(void *obj);
        void *arg;
};

void rcu_read_lock(void);

void rcu_read_unlock(void);
```

In contrast, call_rcu() schedules a callback function func(arg) after a full grace period. Because call_rcu() never sleeps, it may be called with locks held and from interrupt context. The call_rcu() function uses its struct rcu_head argument to remember its callback function and argument during the grace period. An rcu_head is often placed within the structure being protected by RCU, eliminating the need to allocate it separately.

The primitives rcu_read_lock() and rcu_read_unlock() demark a read-side RCU critical section but generate no code in non-preemptive kernels. In preemptive kernels, they disable preemption, thereby preventing preemption from prematurely ending a grace period. RCU-like mechanisms also may be used in preemptive kernels without suppressing preemption, as demonstrated by the K42 research OS.

Most modern microprocessors feature weak memory-consistency models, which require special memory-barrier instructions. However, these instructions are difficult to understand and even more difficult to test, so the Linux list-manipulation API is extended to include memory barriers, as suggested by Manfred Spraul and as shown in Listing 2. The hlist primitives recently were added by Andi Kleen in order to reduce memory requirements for large hash tables.

Listing 2. RCU Extensions to the Linux List-Manipulation API

```
list_add_rcu(struct list_head *new,
             struct list_head *head);
list_add_rcu_tail(struct list_head *new,
                  struct list_head *head);
list_del_rcu(struct list_head *entry);
list_for_each_rcu(struct list_head *pos,
                  struct list_head *head);
list_for_each_safe_rcu(struct list_head *pos,
                       struct list_head *n,
                       struct list_head *head);
list_for_each_entry_rcu(struct list_head *pos,
                        struct list_head *n,
                        struct list_head *head);
list_for_each_continue_rcu(struct list_head *pos,
                           struct list_head *head);
hlist_del_rcu(struct hlist_node *n);
void hlist_del_init(struct hlist_node *n);
hlist_add_head_rcu(struct hlist_node *n,
                   struct hlist_head *h);
```

When RCU is applied to data structures other than lists, memory-barrier instructions must be specified explicitly. For an example, see Mingming Cao's RCU-based modifications to System V IPC.

How to Use RCU

Although RCU has been used in many interesting and surprising ways, one of the most straightforward uses is as a replacement for reader-writer locking. In fact, RCU may be thought of as the next step in a progression. The rwlock primitives allow readers to run in parallel with each other, but not in parallel with updaters. RCU, on the other hand, allows readers to run in parallel both with each other and with updaters.

This section presents the analogy between rwlock and RCU, protecting the simple doubly linked list data structure shown in Listing 3 with reader-writer locks and then with RCU. This structure has a struct list_head, a search key, a single integer for data and a struct rcu_head.

Listing 3. A Data Structure Protected by RCU

```
struct el {
     struct list_head list;
     long key;
     long data;
     struct rcu_head my_rcu_head;
};
```

The reader-writer-lock/RCU analogy substitutes primitives as shown in Table 2. The asterisked primitives are no-ops in non-preemptible kernels; in preemptible kernels, they suppress preemption, which is an extremely cheap operation on the local task structure. Because rcu_read_lock() does not block interrupt contexts, it is necessary to add primitives for this purpose where needed. For example, read_lock_irqsave must become rcu_read_lock(), followed by local_irq_save().

Table 2. Reader-Writer Lock and RCU Primitives

Reader-Writer Lock	Read-Copy Update
rwlock_t	spinlock_t
read_lock()	rcu_read_lock() *
read_unlock()	rcu_read_unlock() *
write_lock()	spin_lock()
write_unlock()	spin_unlock()
list_add()	list_add_rcu()
list_add_tail()	list_add_tail_rcu()
list_del()	list_del_rcu()
list_for_each()	list_for_each_rcu()

Table 3 illustrates this transformation for some simple linked-list operations. Notice that line 10 of the rwlock delete() function is replaced with a call_rcu() that delays the invocation of my_free() until the end of a grace period. The rest of the functions are transformed in a straightforward fashion, as indicated in Table 2.

Although this analogy can be quite compelling and useful—in Dipankar Sarma's and Maneesh Soni's use of RCU in dcache, for example—there are some caveats:

- Read-side critical sections may see stale data that has been removed from the list but not yet freed. There are some situations where this is not a problem, such as routing tables for best-effort protocols. In other situations, such stale data may be detected and ignored, as happens in the 2.5 kernel's System V IPC implementation.
- Read-side critical sections may run concurrently with write-side critical sections.
- The grace period delays the freeing of memory, which means the memory footprint is somewhat larger when using RCU than it is when using reader-writer locking.
- When changing to RCU, write-side reader-writer locking code that modifies list elements in place often must be restructured to prevent read-side RCU code from seeing the data in an inconsistent state. In many cases, this restructuring is quite straightforward; for example, you could

create a new list element with the desired state, then replace the old element with the new one.

Where it applies, this analogy can deliver full parallelism with hardly any increase in complexity.

RCU Synchronizing with NMIs

Retrofitting existing code with RCU as shown above can produce significant performance gains. The best results, of course, are obtained by designing RCU into the algorithms and code from the start.

The i386 oprofile code contains an excellent example of designed-in RCU. This code can use NMIs (nonmaskable interrupts) to handle profiling independently of the normal clock interrupt, which permits profiling of the clock interrupt handler. Synchronizing with NMIs traditionally has been difficult; by definition, no way exists to block an NMI. Straightforward locking designs therefore are subject to deadlock, where the CPU holding the lock receives an NMI, and the NMI handler spins forever on this same lock. Another approach is to mask NMIs in software using things like spin_trylock(). This method, however, produces cache bouncing and memory-barrier overhead, and the NMIs thus masked are lost. The solution in nmi_timer_int.c is as shown in Listing 4.

Listing 4. Using RCU in nmi_timer_int.c

```
static void timer_stop(void)
{
        enable_timer_nmi_watchdog();
        unset_nmi_callback();
        synchronize_kernel();
}

static struct oprofile_operations nmi_timer_ops = {
        .start  = timer_start,
        .stop   = timer_stop,
        .cpu_type = "timer"
};
```

The synchronize_kernel() ensures that any NMI handlers executing the old NMI callback upon entry to timer_stop() have completed before timer_stop() returns. The code for oprofile_stop() and oprofile_shutdown() shown in Listing 5 illustrates why this is important. Notice that oprofile_ops->stop() invokes timer_stop(). Therefore, if oprofile_stop() and oprofile_shutdown() were called in quick succession, the newly freed CPU buffers could be accessed by an ongoing NMI. This action could surprise any code quickly reallocating this memory.

Listing 5. More Code from nmi_timer_int.c

```
void oprofile_stop(void)
{
        down(&start_sem);
        if (!oprofile_started)
                goto out;
        oprofile_ops->stop();
        oprofile_started = 0;
        /* wake up the daemon to read remainder */
        wake_up_buffer_waiter();
out:
        up(&start_sem);
}

void oprofile_shutdown(void)
{
        down(&start_sem);
        sync_stop();
        if (oprofile_ops->shutdown)
                oprofile_ops->shutdown();
        is_setup = 0;
        free_event_buffer();
        free_cpu_buffers();
        up(&start_sem);
}
```

Use of RCU eliminates this race naturally, without incurring any locking or memory-barrier overhead.

Incremental Use of RCU

Using RCU is not an all-or-nothing affair. It may be applied incrementally to particular code paths as needed. A good example of this is a patch coded by Dipankar Sarma that prevents ls /proc from blocking fork(). The changes are as follows:

1. The read_lock() and read_unlock() of tasklist_lock in get_pid_list() are replaced by rcu_read_lock() and rcu_read_unlock(), respectively.
2. A struct rcu_head is added to task_struct in order to track the task structures waiting for a grace period to expire.
3. The put_task_struct() macro invokes __put_task_struct() via call_rcu() rather than directly. This ensures that all concurrently executing get_pid_list() invocations complete before any task structures that they might have been referencing are freed.
4. The SET_LINKS() and REMOVE_LINKS() macros use the _rcu form of the list-manipulation primitives.
5. The for_each_process() macro gets a read_barrier_depends() to make this code safe for the DEC Alpha.

The problem is get_pid_list() traverses the entire tasklist in order to build the PID list needed by ls /proc. It read-holds tasklist_lock during this traversal and blocks updates to the tasklist, such as those performed by fork(). On machines with large numbers of tasks, this can cause severe difficulties, particularly given multiple instances of certain performance-monitoring tools.

Dipankar's modifications are shown in Table 4, changing only two files, adding 13 lines and deleting seven for a six-line net addition to the kernel. This patch does delete a pair of tasklist_lock uses, but none of the other 249 uses of tasklist_lock are modified. This example demonstrates use of RCU for a late-in-cycle optimization.

Where Do We Go from Here?

RCU will become more important as CPU architecture continues to evolve. Nonetheless, other primitives always will be needed. It is quite likely that Rusty Russell's implementation of RCU (the call_rcu() and synchronize_kernel() primitives themselves) can be modified to be entirely free of locks, memory barriers and atomic instructions. This implementation might run faster than the current 2.5 kernel implementation.

Numerous people are looking at new uses of RCU in the VFS layer, VM code, filesystems and networking code. I look forward to continuing to learn about RCU and its uses and am grateful to the many people who have tried it out.

Paul E. McKenney has worked on SMP and NUMA algorithms for longer than he cares to admit. Prior to that, he worked on packet-radio and Internet protocols (but long before the Internet became popular). His hobbies include running and the usual house-wife-and-kids habit. This work represents the view of the author and does not necessarily represent the view of IBM.

Archive Index Issue Table of Contents

Advanced search

Advanced search

At the Forge

Bricolage Alerts

Reuven M. Lerner

Issue #114, October 2003

Keep track of what's happening on your Web site with alerts for key events.

Last month, we started to look into the powerful open-source Bricolage content management system (CMS) written by David Wheeler and based on mod_perl, HTML::Mason and PostgreSQL. Content management software is a relatively new type of Web application, designed to make it possible to manage large Web sites. Bricolage has been widely hailed as a new open-source success story, demonstrating that proprietary software is not necessarily more flexible or more powerful than its free software counterparts.

It's important to remember that although a CMS is a Web-based application that depends on dynamically generated server-side programs, the output from a CMS typically is a static site. So even though Bricolage is a large Web/database program and customizing Bricolage requires server-side programming skills, it actually is an application meant to be used on a day-to-day basis by nonprogrammers. Indeed, organizations from Salon.com to *MacWorld* on-line currently use Bricolage. And thousands of other Web sites, from CNet to LinuxJournal.com, use content management solutions ranging from Vignette (complex, proprietary and expensive) to PHPNuke (simple, open source and free of charge).

This month, we look at one of my favorite Bricolage features, alerts, which allow users to keep track of different activities on a Bricolage system. Not only do alerts keep us informed of what is happening, they also provide a good way to see how the system works—by looking at the list of objects, by the actions that can be taken on each of those objects and by knowing when the alerts actually are triggered.

Alerts

Bricolage, like most CMS software, moves articles through stations in a pipeline. In Bricolage, these stations are known as desks, reflecting the software's origins in the world of journalism. Stories thus begin at the edit desk, move to the copy desk, then to the legal desk and finally to the publish desk, from which they actually can be published on the Web.

On a small Web site being managed by only a handful of people, it is easy to keep track of which articles are kept where. But once you get beyond a small number of people or a small number of articles, it becomes difficult to keep track of what is happening with each article.

One way to handle this work flow is to look at the various desks, one at a time, to see what stories are on each one and then take appropriate action. But this looking can get tedious, and you might want to track stories in the news category or those by a particular author, rather than all of them. Moreover, it would be nice to receive notification of work-flow events via e-mail.

This functionality exists in Bricolage, and in true open-source fashion, it is customizable to an amazing degree. To create or modify alerts, click on the alert types menu item under the system menu in the lower right-hand side of the screen. This is one of the admin menus, meaning it's accessible only to users with administrative privileges. The resulting screen, like most other administrative screens in Bricolage, allows you to search for an alert by name or view all alerts that begin with a particular letter.

Creating Alerts

You can create a new alert type by clicking on the create new alert type link. This brings up a short HTML form that asks you to identify the Bricolage object on which you want to put an alert. So if you want to be alerted when something happens to stories, ask for alerts on stories. And if you want to be alerted when a new user is added to the system, ask for alerts on users. In short, almost any object in Bricolage can be placed under an alert.

As an example, let's create an alert to tell us when any story with “Linux” in the headline is moved from one desk to another. Once again, Bricolage makes it easy for us to monitor any object. This, however, is undoubtedly one of the more common types of alert that take place.

We now choose create new alert type from the admin menu, then create a new alert on Story objects. We then are presented with a list of actions that Bricolage can watch for us, ranging from category added to story to story published to element deleted from story. For this example, we choose Story moved to desk and name it Linux story moved.

Each alert type must have an owner; in this particular case, the owner is me, because I'm logged in as myself. The only user initially created by Bricolage, whose login is admin, is known as Bricolage Administrator. This admin user should be treated the same as the UNIX root user. It certainly can own alerts, but you are better off creating additional users (with the admin/user menu in the bottom left-hand corner), giving yourself administrative permissions and then logging in as yourself rather than as the administrator.

In any event, clicking on the next button at the bottom of this page brings you to the main alert type editing screen, which is used to create new alert types and modify existing ones. Each alert has four parts:

- Properties: the name and owner of the alert type, which we entered on the previous page.
- Rules: a description of when the alert should fire. Each rule consists of a variable (selected from a pull-down menu, thus avoiding the potential for misspellings), a comparison test and a text field into which you can enter the comparison value.
- Content: the e-mail message sent to alert recipients. This can include a number of different variables, from the story's headline to its publication date.
- Recipients: the users and groups who should receive the alert, if and when it fires. You can send an alert to all editors, to all authors or to George and Frank but not Deborah and Mary.

Alert Rules

The rules are perhaps the most interesting part of the Bricolage alert system because of the relative ease with which nonprogrammers can create and edit them. However, there is the potential for danger with the =~ and !~ operators, which Perl programmers should recognize as indicators of regular expressions. This clearly is a double-edged sword, because regular expressions can be enormously powerful to the enlightened and extremely dangerous (and frustrating) to the ignorant.

So, we can create our alert by choosing Story title from the attribute list on the left side, =~ from the comparison operator list and entering Linux as the value in the text field, giving us:

```
Story title =~ Linux
```

We have to use =~ and not =, because we want to look for Linux in any part of the title rather than match the entire title. If we were interested in either Linux or Perl, we would search for:

```
Story title =~ Linux|Perl
```

Experienced Perl hackers might be pleasantly surprised to learn that =~ and !~ are case-insensitive here.

The subject and text of the alert can contain any text you wish, including interpolated Perl variables that Bricolage has defined for us. In a nice use of JavaScript, Bricolage lets you choose variables from a selection list, avoiding the possibility of typos and other errors. Also, this is a handy reference so you don't need to remember all of the variable names available when working with stories. In this manner, we can set the subject of the alert to be:

```
The story $title was just moved to $desk
```

The body of the alert message then contains a different message, but it also can contain variables. For example, you can set the alert to read:

```
You asked to be notified when Linux-related articles
are moved to a new desk. Well, $trig_full_name
just moved "$title" to the $desk
desk. I hope you're happy now.
```

Receiving Alerts

When an alert fires, its message is sent to all of the designated recipients in two different ways. An e-mail message is sent to each of the registered users, informing them, with the message we defined above, that the change has taken place. But Bricolage also keeps track of these items within its database, making it easy to keep track of alerts over the Web. For example, if an alert is fired from the Linux-related item we described above, we receive an indication by e-mail. But we also can see a summary of all the alerts we have received on the my alerts page, which you can get to by clicking on the my alerts button located at the top of each screen.

Alerts stay on this screen until they are acknowledged. The alert screen is not meant to be a long-term storage system; rather, it is a simple messaging agent that allows an editor to look at all of the relevant things that have happened to the site in the recent past.

You can acknowledge, and thus remove, an alert from the my alerts page by clicking in one or more check boxes and then selecting the acknowledge checked button at the bottom of the list. You also can acknowledge all of them at once, without having to use check boxes, by clicking on the acknowledge all button.

If I were the editor of a medium-sized Web site, I would spend a great deal of time defining alerts that would let me know when important events had occurred, such as when writers sent items to my desk.

Learning from Alerts

Alerts are an excellent and practical way for Bricolage to bring relevant information to the appropriate user, rather than forcing users to go and seek out the information. But a side benefit of these alerts is they help new Bricolage administrators and programmers understand the different actions that can occur on different objects in the system. A simple example is the user object: we can create an alert that notifies us whenever users are created or deactivated, change their settings or change their password. This sort of alert is clearly uninteresting for editors but can be of supreme importance to system administrators.

A more complex example is the template object, which will be discussed next month. Templates define the ways in which stories are displayed, so it's important to keep track of them and any modifications made to them. Therefore, more items are associated with templates than with users—you can keep track of when a template is deployed, when it is edited and even when it is moved to its desk. Indeed, if you didn't previously understand that templates have their own desk, just like stories, the alert system would have made that clear to you.

I personally have learned quite a bit about Bricolage from poking through the system from different angles, including alert definitions. If you are new to Bricolage and haven't quite figured out how everything works, even after reading the documentation and expanding all of the menus (which is vital if you are to see where the different parts of a Bricolage site are defined and who has permission to set them), looking through the alert definitions makes things clearer. Moreover, if you are unsure of how a particular object is handled in the system, you always can create and register for an alert on that object, disabling it once you learn how it works.

Conclusion

As we saw last month, Bricolage is a large and complex system. Alerts make it possible for relevant information to pursue you, freeing you from having to hunt it down on a number of different screens. Although alerts are only a small part of the Bricolage system, they offer an intriguing view of its depth and breadth and of the options available with different objects.

Resources

The main source of information about Bricolage is the project's Web site, at bricolage.cc. That site has pointers to downloadable source code (hosted at SourceForge), documentation and an instance of Bugzilla (bugzilla.bricolage.cc) for bug reports and feature requests.

There are several Bricolage mailing lists, hosted by SourceForge, in which the developers participate actively. If you have questions or want to learn about new releases, you can subscribe from the SourceForge page (sourceforge.net/projects/bricolage).

The Bricolage documentation generally is quite good, though technical. A more user-level introduction to the system was published by O'Reilly and Associates as an appendix to the recently published book about Mason. You can read that appendix on-line at www.masonbook.com/book/appendix-d.mhtml.

For more information about PostgreSQL, see the project's main site at www.postgresql.org. For more information about Apache, see httpd.apache.org. To learn more about mod_perl, look at perl.apache.org. Remember that Apache 2.x and mod_perl 2.x both are unsuitable for Bricolage, although that may change by the time you read this. You can learn more about Mason from the Mason book site (www.masonbook.com) and from the Mason home page (www.masonhq.com).

Finally, you can learn more about David Wheeler (the primary author and maintainer of Bricolage) at david.wheeler.net and about his company Kineticode at www.kineticode.com.

Reuven M. Lerner (reuven@lerner.co.il) is a consultant specializing in open-source Web/database technologies. He and his wife, Shira, recently celebrated the birth of their second daughter, Shikma Bruria. Reuven's book *Core Perl* was published by Prentice Hall in early 2002, and a second book about open-source Web technologies will be published by Apress in 2003.

Archive Index Issue Table of Contents

Advanced search

Cooking with Linux

Mirror, Mirror, of It All

Marcel Gagné

Issue #114, October 2003

Make simple backups and keep every copy of your Web or FTP site up to date with some standard tools that probably are already on your system.

François, what are you doing? When I asked you to mirror our Web sites, I did not mean that you should hold a mirror up to the screen. You can be very silly, *mon ami*. What I meant was that you should make a copy of our Web sites onto that other machine. François, what are you looking at? Ah, our guests have arrived! Why did you not tell me? Welcome, *mes amis*, to *Chez Marcel*, home of fine Linux fare and exceptional wines.

Speaking of wine, François! To the wine cellar, *immédiatement*. Please bring back the 1999 California Stag's Leap District Cabernet Sauvignon. This bold, smooth wine is the perfect mirror to today's menu. As you know, *mes amis*, the theme of this issue is system administration. On today's menu, we are going to sample a number of alternatives for mirroring data. The reasons for mirroring data are many. The obvious first reason is the not altogether sexy but extremely important subject of backups. Other reasons include creating mirrors of FTP sites for local network updates, such as your own RPM update repository, or mirroring Web sites for fast, off-line reading.

Many people who do regular backups are doing them to a disk on one of their other machines. Others still are backing up to a second disk on the same machine. Given that an extra hard drive added to a system is extremely inexpensive these days and high-capacity tape drives can cost substantially more, it isn't that unusual to find this kind of solution being used.

Backing up from one disk to the other, or creating a mirror of your data, can be as simple as doing a recursive copy using **cp**. For instance, if I wanted to copy everything in my home directory to a second disk with a lot of space, I might do the following:

```
cp -rfupv /home/mgagne /disk2/
```

As you probably expect, the -r option indicates a recursive copy (all the subdirectories), and the -v tells the command to be verbose. Because I don't want to be warned about each file being overwritten, I add -f to force the copy; the -p ensures that permissions are saved properly as well. Finally, the -u option tells the cp command to copy only files that have been updated. This speeds up the process on subsequent copies.

It all works very well, but copying from machine to machine requires a few extra steps. With your Linux system, you actually have a lot of tools at your disposal beyond the humble cp. For starters, if you want to copy or back up an entire Web site, try the **wget** command, originally written by Hrvoje Niksic:

```
wget -m http://www.websitename.dom
```

Starting at the top of your chosen Web site, wget walks through the entire site, saving all appropriate HTML files and images. The -m in this case means mirror, but it also encompasses several other options, specifically -r, -N, -l inf and -nr. These options tell wget to do a recursive fetch, turn on timestamps, allow for an infinite number of levels and not to remove the FTP directory .listing files, respectively.

All files on the Web site are saved in a local directory with the same name as the Web site. In the example above, that would be www.websitename.dom. Add a new file to your Web server, run the command again and only that new file is transferred, thus making the job of keeping things up to date that much faster.

This is a great tool for its intended purpose, but its primary function is to deal with Web sites. It is possible, however, to use wget to download from FTP servers as well. If you are transferring from anonymous sites, the format is almost identical to the one used to mirror a Web site:

```
wget -m ftp://ftp.ftpsitename.dom
```

If, on the other hand, you want to back up a user directory where a user name and password are required, you need to be a little fancier:

```
wget -m ftp://username:password@ftp.sitename.dom
```

This approach has a couple downsides. First, your password is sent across the network in plain text, which may not be a big deal depending on how much you trust your network. In a pinch, you could do a recursive secure copy with the **scp** command. Because scp is part of OpenSSH, you have the advantage of knowing that you are using secure, encrypted file transfers. Pretend that you want to copy your whole Web site, starting from the Apache server root. It would look something like this:

```
scp -rpv /var/www root@remote_host:/mnt/backupdir
```

The -r indicates a recursive copy, and the -p tells scp to preserve modification times, ownership and permissions from the original files and directories. If you are transferring large amounts of data, you might consider using the -C option, which does compression on the fly. It can make a substantial difference in throughput.

Possibly the biggest problem with all these methods of mirroring data is it can take a great deal of time. wget will download new files from an FTP server, but there is no option to keep a directory entirely in sync by deleting files. Secure copy is nice, but it doesn't have any mechanism for transferring only files that are changed. That's the second downside. Making sure that the data stays in sync without transferring every single file and directory requires a program with a bit more finesse.

The best program I know for this is probably Andrew Tridgell's rsync. *Linux Journal*'s own Mick Bauer did a superb job of covering this package in the March and April 2003 issues of this fine magazine, so I won't go over it again other than to say you might want to look up his two-parter on the subject.

ftpcopy

In many cases, that leaves us with our old friend, FTP—well, sort of. On one side (the machine you want to mirror), you would use your FTP server, whether it was ProFTPD or wu-ftpd. On the other side, you would use Uwe Ohse's **ftpcopy** program. ftpcopy is a fast, easy-to-set-up and easy-to-use program that does a nice job of copying entire directory hierarchies. As it copies, it maintains permissions and modification dates and times, and it does it fast. Furthermore, it keeps track of files that already have been downloaded. This is handy because the next time you run ftpcopy, it transfers only those files that have been changed, thus making your backup even faster.

Some distributions come with ftpcopy, but for the latest version of ftpcopy, go to www.ohse.de/uwe/ftpcopy/ftpcopy.html to pick up the download. Building the package is easy and takes only a few steps:

```
tar -xzvf ftpcopy-0.6.2.tar.gz
cd web/ftpcopy-0.6.2
make
```

In the directory called command, you'll find three binaries: ftpcopy, ftpcp and ftpls. You can run it from here or copy the three files to /usr/local/bin or somewhere else in your $PATH.

Here's how it works. Let's say I wanted to mirror or back up my home directory on a remote system. A basic ftpcopy command looks something like this:

```
ftpcopy -u marcel -p secr3t! \
remote.hostname /home/marcel /mirdir/
```

The -u and -p options are obviously for my user name and (fake) password on the remote system. What follows is the path to the directory you want to copy and then the local directory where this directory structure will be re-created. As the download progresses, you will see something like this:

```
/mirdir/scripts/backup.log: download successful
/mirdir/scripts/checkhosts.pl: download successful
/mirdir/scripts/ftplogin.msg: download successful
/mirdir/scripts/gettime.pl: download successful
```

If you want a little more information on your download, add the --bps option. The results then report the rate of data transfer in bytes per second.

You should consider running ftpcopy with the --help option at least once, and you should be aware of some options. For instance, -s deals with symbolic links, and -l lets you increase the level of logging. If you want to set mirroring to run by means of a cron job, you might want to set logging to 0. Another useful option is -n. If a file is deleted on the remote side, it also will be deleted locally when you run ftpcopy. If you truly are trying to keep systems in sync, this is what you would want. To override this behavior, add -n and no deletes will occur.

Well, *mes amis*, the hour has arrived, and we must all go to our respective homes. Still, it is early enough for a final glass of wine, *non*? François, *mon ami*, if you will do the honors—in fact, make it two glasses, one to mirror the other, *non*? Until next time, *mes amis*, let us all drink to one another's health. *A vôtre santé Bon appétit!*!

Resources

ftpcopy: www.ohse.de/uwe/ftpcopy/ftpcopy.html

OpenSSH: www.openssh.org

rsync: rsync.samba.org

wget: wget.sunsite.dk

Marcel's Wine Page: www.marcelgagne.com/wine.html

Marcel Gagné (mggagne@salmar.com) lives in Mississauga, Ontario. He is the author of the newly published *Moving to Linux: Kiss the Blue Screen of Death Goodbye!* (ISBN 0-321-15998-5) from Addison-Wesley. His first book is the highly acclaimed *Linux System Administration: A User's Guide* (ISBN 0-201-71934-7). In real life, he is president of Salmar Consulting, Inc., a systems integration and network consulting firm.

Advanced search

EOF

Nation without Software: Why Iraq Needs Linux

Ashraf T. Hasson

Issue #114, October 2003

An ignorant, corrupt government, war and sanctions have left Iraq without a software industry, providing a blank canvas for Linux.

Iraq has been in the dark ages of software, both in terms of using different available software and in developing its own. While the rest of the world was moving toward a major evolution in software usage and development, Iraq fell behind. Because of the crises of wars and sanctions, the largest institutions, both private and public, including the government, didn't have the opportunity to research and implement the best software for their needs. Instead, they took the first proposal to come along; unfortunately, Linux was not the proposal.

The software wheel turned too quickly to follow, especially due to the absence of the types of facilities needed to enable someone to become productive and successful in developing software.

With the new situation in Iraq, copyright has returned, and companies are willing to enforce infringed rights for unauthorized software usage, or at least prevent it from happening again, especially once things settle down here. This will push sites in Iraq to look for a substitute for proprietary software, because cost plays a definite role here, although it is not the main consideration.

Although it's nearly impossible to live without a computer these days, the ignorance and corruption of the past Iraqi ruling government held back growth and even caused the degradation of software usage in many vital departments. Hence, while things are still fresh, Linux could play a major role in aiding the rebuilding of a real infrastructure in several different fields without incurring heavy expenses.

This may require a Linux Community Center, which could handle some difficult techniques, to be established in Iraq to help with taking the first steps. You can imagine the whole situation as a white canvas to be sketched on and filled in with colors, freely choosing which colors to use.

I'm sure there are unlimited ideas to be implemented in many fields, which would have great benefits for all in aiding different aspects of life. As we are talking about infrastructure, the education sector seems to be one of the most suitable spots for Linux adoption, which might make it possible for easy, fast and professional implementation of Linux to do the job here in Iraq.

Establishing or rebuilding the knowledge network itself may take some time, which in turn reveals another angle in making Linux one of the main stones in this construction—making it part of daily professional computer usage. For instance, until now many high schools have not changed their syllabi since the last decade, and they do not even teach hardware or software basics.

Here it could be possible to teach Linux OS basics not in terms of migration from one OS to another, or even in terms of a new OS, but rather in terms of a powerful OS that will give fresh brains a chance to choose before getting locked into something else. This also applies to some colleges and institutions on a more advanced level. There is a lot of work to be done.

At the new Linux Users Group of Iraq, our first priority is to establish a Linux training lab and a lending library. We are asking both Al-Nahrain University in Baghdad and Baghdad University to host Linux training classes. We invite key information technology professionals from important public and private institutions in Iraq to participate in Linux training classes and seminars. As our members' institutions develop their Linux installation plans, we will collaborate on development of localized documentation and shared resources, including mailing lists and distribution mirrors.

Our Web site at linux-iraq.org will be updated with information on how you can help.

Ashraf T. Hasson lives in Baghdad, Iraq. He is a graduate of the Laser Department of the College of Engineering and is now working on a Master's of Science degree in Laser Engineering. He is the founder of the new Linux Users Group of Iraq.

NEC Fault-Tolerant Linux Server

Dan Wilder

Issue #114, October 2003

Product Information.

- Manufacturer: NEC Corporation
- URL: www.necft.com
- Price: $24,000+ US

The Good.

- Solid construction.
- No-tools, hot-swap redundant CPU, disk, power supply and PCI modules.
- Auto-isolation of failed hardware.

The Bad.

- Not blazingly fast.
- Kernel 2.4.2 may not fit all needs.
- Ships with old versions of dæmons containing known security issues.

NEC Corporation's Express5800/320La is the first commercially available general-purpose server offering hardware fault tolerance for Linux. Intended for standalone use or as an element in a high-availability cluster, this server features redundant CPUs, memory, disk, I/O and power. Hardware failover circuitry allows normal operation to continue despite loss of any single unit. Hot-swap capability extends beyond the usual power supply and disk. If a CPU, RAM or I/O card fails on this system, it is isolated and processing continues without interruption. You may replace the failed item at your convenience, without taking the entire system down. This could provide significant cost savings, for example, to a company needing servers that are always up, at far-flung locations where technical support might be hours away. Applications require no high-availability modifications to use this system as a standalone server, nor do they require failover scripts and planning.

Thousands of these servers have shipped with other operating systems, and now Linux is available on them. A stock Linux kernel provides too little error detection and recovery for this mode of operation, so NEC has added extensive hardening. SCSI, Ethernet and Fibre Channel drivers and support code in particular are modified to provide fault detection and failover. NEC's currently shipping kernel is based on version 2.4.2, with backports of some later changes. At the time of this writing, NEC was reviewing and documenting its kernel changes for a planned public release, perhaps through OSDL's Carrier Grade Linux Project. NEC is a founding member and a sponsor of OSDL.

Features

The Express5800/320La has four Pentium III 800MHz processors arranged in pairs together with RAM and other circuitry, in two hot-swappable CPU modules. Both modules run the same instructions in lockstep, checking each other's outputs. A failed unit is isolated almost instantly, allowing processing to continue with no observable interruption. Monitoring software keeps tally of recoverable failures, such as ECC corrections to memory output, allowing diagnosis of certain incipient problems prior to larger failures. The stock filesystem on this server is ext2.

A total of three pairs of internal 18, 36 or 73GB drives may be installed and configured in RAID-1 pairs, providing up to 219GB of internal storage. An NEC S1200 RAID array may be connected through a redundant Fibre Channel, providing up to 2TB of additional fault-tolerant storage.

Two PCI modules feature dual identical sets of PCI cards. The base unit has one Ethernet card in each module. Both cards are connected to the same network; when one fails, the other takes over using the same MAC and IP addresses. All modules and power supplies plug in to a passive backplane.

Hardware watchdog timers look for system failure—for example, a system lockup due to kernel panic—and may be configured to initiate an automatic reboot either to full run mode or to diagnostic mode.

This server is large, measuring 14“ wide by 21.5” high by 27.5“ deep and weighing about 150 pounds. An 8U rackmount version also is available. A three-year warranty is included. Telephone support is provided by NEC during regular business hours.

Unpacking and Startup

Unpacking our review unit's well-traveled shipping crate, I observed a warning sticker on the case saying “Exercise caution when handling the system to avoid personal injuries.” NEC isn't kidding. The help of a strong coworker was needed to lift this thing gently out of its shipping crate and place it on the floor. Our demo unit had dual Seagate ST318404LC 18G SCSI drives, 1GB of RAM and two Ethernet cards.

Internal assemblies look to be well made, with no tools required for removal and replacement. Better labeling of the units would be nice, though. Fans are located in the removable units, so you don't have to take one of these servers down to replace a failing fan. Even the power cords are redundant. This allows powering the server from two independent power sources, not to mention letting the harried system administrator unplug a cord to untangle it without interrupting anything.

After pressing the power switch, located under a hinged plastic protective lid, a chorus of cooling fans kicked in with a hearty whoosh measuring 63 dBA at the front panel, 74 dBA at the back. The front panel LCD status monitor showed diagnostic messages and LEDs flashed. After about two minutes, the system completed a power-on self-test and booted up into NEC Linux, which is based on Red Hat Linux 7.1.

The popular bonnie++ disk test program was the first thing we tried on this system. Immediately upon bonnie++ startup, the fault light on one CPU module came on. The test completed, as expected, but it seemed prudent to correct the problem with the server. An NEC engineer reached over the support line had us run a few tests, and then suggested that the passive backplane had suffered mechanical damage, possibly in shipping. The backplane isn't hot-swappable. He wanted to examine it, so we arranged an exchange of servers. The new server arrived in good time, booted up and survived bonnie++ quite nicely.

To test networking recovery, I unplugged the Ethernet cables from each of the two Ethernet cards, one at a time. Ping indicated a few packets were lost, but overall communication was maintained. An rsync between the test unit and another server completed without error, despite continual unplugging of alternate cables, one at a time, with several seconds of overlap while both were plugged in.

While running bonnie++, I disconnected power to each CPU module and then reconnected it. In each case the CPU module came back up after running diagnostics for a couple of minutes. The disk benchmark results were unaffected.

Benchmarks

The best benchmark is the load you plan to run, and a lot of benchmarks can be used. Each captures some limited view of what a system can do. As benchmarks become popular, manufacturers tweak hardware and installations to optimize benchmark results, and results thus become less applicable.

At *Linux Journal* we perform a rough evaluation of servers based on results of bonnie++, kernel build and the PostgreSQL regression test. These are run several times and the results averaged. We compared this server to a spare generic server having a single Athlon XP 2100+ processor using two different disk configurations, a single IDE drive and IDE hardware RAID-5. The results are shown in Tables 1–5.

The NEC unit held up well against the IDE RAID system in I/O tests, beating it handily in block output and block rewrite and narrowly in block input. In creating large numbers of zero-size files, the generic machine having the faster CPU won. The NEC server fell between the two configurations of the generic machine on elapsed time in the PostgreSQL regression test, while devoting a higher proportion of its CPU time, though at a lower load average—a measure of number of processes ready to run but not running. The kernel build test saw the NEC unit fare not nearly so well. Perhaps this is because compiling a kernel is computationally more intensive than the other two tests, but it doesn't adapt as well to multiple CPU operation, even though multiple processes are at work.

We did dry runs of these tests, and the relative loads on the two CPUs swung widely back and forth during the bonnie++ and kernel compilation tests. The PostgreSQL test showed fairly stable load allocation between the two CPUs. This is not entirely surprising, as the PostgreSQL test splits processing between client and server, offering better possibilities for distributing load between two CPUs.

Overall, the NEC machine was not blazingly fast as compared with our generic computer, a middle-of-the-road machine of recent vintage. But, it held its own nicely. The NEC machine's claim to fame, in any case, is its fault tolerance. In this, our generic machine offers not much comparison.

Software

The NEC server comes equipped with NEC Linux, derived from Red Hat 7.1. It is text-only at the console, with no X server included. X libraries and clients are furnished for use from remote displays. Installation was reasonably complete, although I found somewhat annoying the absence of certain programs I rely on, such as procinfo.

Software re-installation is accomplished using Red Hat's kickstart method. When we tried it, re-installation from the provided CDs went without a hitch.

Some of the installed dæmons are very old releases and contain serious security holes. Examples include Sendmail 8.11.2, Apache 1.3.19, OpenSSL 0.9.6 and OpenSSH 2.5.2p2. We were unable to learn of concrete plans to ship newer versions. John Fitzsimmons of Aspire Communications, who served as our liaison with NEC during this review, indicated NEC expects its customers will customize the distribution to their own liking, clearing the final setup with NEC prior to deployment. Upgrading these things is likely to be the least part of the customization. Notwithstanding, an upgraded NEC Linux may be available later this year. It may contain an X server; we hope it contains security upgrades.

NEC supplies extensive proprietary software for configuration and monitoring of the server. This allows setup and configuration of the redundant hardware and monitoring and reporting over SNMP or other means to remote systems.

Overall

At press time, quite a bit of work remained to be done on the software load. For the price, starting at $24,000 US for the server reviewed, it would be nice if the customer did not immediately need to begin significant security-related upgrades.

We have some concern about the future of the kernel on which this server's fault tolerance depends. There have been many changes to the 2.4 kernel since 2.4.2, the version this server's kernel is based on. We were unable to obtain a clear reading of NEC's plans with respect to carrying forward its changes to later 2.4. Although there is a plan to carry forward these changes to kernel version 2.6 and beyond, and to offer Linux on other NEC fault-tolerant servers, there seems to be no announced dates associated with this plan.

NEC told us they are firmly committed to releasing all changes to the public under GPL. Until this happens, some time after this article goes to press, there is no way to evaluate how extensive NEC's changes are or what their likely fate might be, as to integration into the mainline sources for the kernel.

Our overall evaluation is favorable, but with caution. This server has more than adequate processing capacity for many applications. It is nice to be able to replace failed hardware without bringing down a system. It is nicer still to have hardware that takes itself out of service, without interrupting anything, pending a convenient time for repairs. That said, we'd advise a company considering purchase of a number of these servers to ask careful questions concerning ongoing kernel development. Also, ask about NEC's demo or “try 'n buy” programs under which you may be able to obtain one of these machines and test it hard against its intended application.

Resources

Fault-Tolerant Linux Summary and Request for White Paper: www.necsolutions-am.com/servers/products. If that doesn't work, go to servers.nectech.com and follow links Resources→Q & A. Then search for “Linux” in the document and follow the “Read More” link.

Open Source Development Lab: www.osdl.org

Dan Wilder is technical manager at Specialized Systems Consultants, Inc.

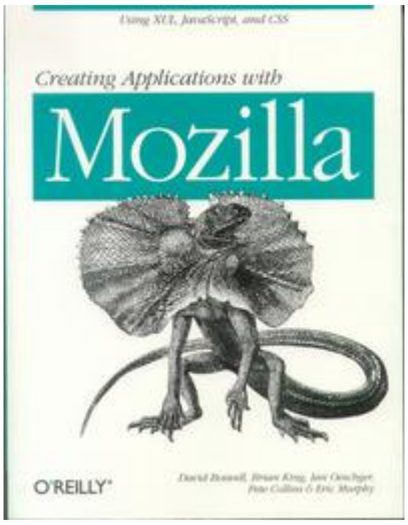
Archive Index Issue Table of Contents

Advanced search

Creating Applications with Mozilla by David Boswell, Brian King, Ian Oeschger, Pete Collins and Eric Murphy

Paul Barry

Issue #114, October 2003



O'Reilly & Associates, Inc., 2002

ISBN: 0-596-00052-9

$39.95 US (hardcover)

Next to Linux, Mozilla is probably the most well-known and widely discussed open-source project. O'Reilly's *Creating Applications with Mozilla* appeared hot on the tail of the official Mozilla 1.0 release. It is notable for a number of reasons: 1) the entire text is available on-line (books.mozdev.org/chapters), as the book was written under the Open Publication License, and 2) it explains why the 1.0 release of Mozilla took so long to make an appearance.

This book shows that Mozilla is much more than a browser. In addition to offering capable e-mail, newsgroup and Web page composer applications, Mozilla contains a sophisticated Web client development environment. A series of technologies allows programmers to interact with and control Mozilla, and at slightly more than 450 tightly packed pages, *Creating Applications with Mozilla* is the definitive reference to these technologies.

The development framework is strong, but the book has some weak points. The title led me to believe the book was of the how-to, tutorial type, so I was a little disappointed to find this was not the case. Some tutorial material is presented, but it is much more of a reference book. Readers should be already familiar with at least JavaScript, CSS and XML. Without a background in these technologies, they may struggle with *Creating Applications with Mozilla*.

I would have welcomed more extensive use of screenshots, especially in the chapters that describe creating and customizing Mozilla GUIs. The authors rely on textual descriptions such as, “the icons appear to the left and right of the bar, while the flexed text panel takes up the remaining space.” Pictures would have been better.

There's also some confusing language, especially in the earlier chapters. A particularly bad example of this is a sentence taken from the discussion of Box Attributes on page 63: “But there are also CSS versions of these properties that use the prefix `box-pack` becomes `box-pack` when it's defined in CSS, for example.” This sort of thing should have been cleaned up during copy editing. However, the entire text (plus errata) is available on-line to allow potential purchasers to try out the book before buying it.

Letters to the Editor

Crash-Proof Penguin

A couple of months ago, you had printed a picture of Tux holding my new “United We Stand” Linux license plate for my car [Letters, *LJ*, February 2003]. After I put the plate on, I had Tux looking out the rear window of the car so he could wave to drivers who were behind me. On Monday, June 30, I was rear-ended by a drunk driver. Everything from the backseat to my rear bumper does not exist anymore. I looked for Tux, but I feared he was lost in all the crumpled metal. When my wife and I went to take pictures of my car, the tow-truck driver asked me if I had a penguin in my car. He handed Tux over to me and said he found him on the side of the road on a patch of grass. He was completely uninjured, not even a speck of dirt on him. I'm impressed that Tux survived the accident. I wish he had given me some warning about the DUI driver. As for me, I walked away with a bump and two scratches on the forehead.

—

Paul Ammann

Miss 2.4

On January 4, 2001, Linus Torvalds released the 2.4 kernel into the wild. On that same day, my daughter Jennifer was born. Two and a half years later, she's learning very young that Linux is a special part of her life.

—

Robert Yannetta

Thanks for the SSH Tip

I just wanted to say thank you for the “Eleven SSH Tricks” article in the August 2003 edition of *Linux Journal*, in particular the tip about port forwarding. I have wanted to have a way to use my company's SMTP server while I am traveling, but for obvious reasons that is not allowed. Now I can do it. Thank you for pointing that out to me. It's one more way that open source makes life better.

—

Greg Willden

Dad, What Was `make dep?`

I thought you'd be glad to see how little Dana is already enjoying *Linux Journal* during her very first holidays, in Croatia. She is literally devouring your articles!

—
Jordi Porta

XML Terminology

I noticed a mistake in the third paragraph of “A Template-Based Approach to XML Parsing in C++” in your June 2003 issue. In it, the author states that “a validating parser scans the XML file and determines if the document is well formed, as specified by either an XML schema or the document type definition (DTD). A nonvalidating parser simply reads the file and ignores the format and layout as specified by either the XML schema or the DTD.” “Well formed” simply means an XML document adheres to the syntax required of all XML documents, such as all beginning tags are matched with end tags, or otherwise properly terminated, while “valid” refers to an XML document that is “well formed” and in addition meets all the criteria set out for its specific contents in a schema or DTD (hence the term “validating parser”). All parsers must ensure XML is “well formed”, but the determination of “validity” is optional.

—
David Gutteridge

John Dubchak replies: You're correct in that I didn't make clear the differences between well formed and validity with respect to XML syntax and documents. As a result, the wording and information provided may have caused confusion for some readers owing to the incorrect distinction. As you've correctly stated, all parsers enforce XML syntax to ensure that they are well formed. Well formed by definition is a document that is capable of being parsed by a parser that conforms to the W3C XML specification. On the other hand, validation is the process of verifying the XML document according to the constraints that are defined in either a document type definition, DTD or an XML schema. In order to parse an XML document it must be well formed. The XML specification clearly defines that a conforming parser will encounter a fatal error when attempting to parse a document that is not well formed. Validation is optional. Thank you very much for the feedback and for taking the time to comment on the article.

More on Content Management, Please

I have been subscribing to *LJ* since 1998, and it is the only subscription I have kept for such a long period. The magazine is excellent. It has had its ups and downs, but the most important thing is that it keeps getting better and better over the years. Keep up the good work! One of my favourite columns is At the Forge. I am not sure what Reuven's plan is, but I would like to know if *LJ* has any intentions to add more about open-source content management systems (CMSes), CMSes in general or to take a closer look at CMSes that support blogging. In my opinion, the topic is hot; there is a huge interest and growing support within the Open Source community for CMSes. It could be worthwhile to write a little bit more about it.

—
Ratko Kovacina

Keep reading for more in Reuven Lerner's ongoing series on open-source CMSes. He covers Bricolage alerts this month on page 12. —Ed.

Doc's Politics

When are you going to get rid of the political and biased editorials of Doc Searls? I was a subscriber two years ago but never renewed my subscription because of him and his political diatribes. A magazine about Linux should be just that, not a personal forum for politics.

—
Matt

LINUX JOURNAL

Advanced search

UpFront

diff -u: What's New in Kernel Development

Zack Brown

Issue #114, October 2003

A new system call, **tgkill()**, has been introduced to handle certain obscure error conditions in which a signal sent to one process may end up going to a completely different process. **Ingo Molnar** implemented the call, and **Linus Torvalds** suggested the name tgkill to correspond to the call's inputs: thread and group. The old call, pthread_kill(), allowed the bug involving signal misplacement and only operated on thread IDs as input.

Eugene Weiss has created **Submount**, a new attempt to support hard disk hot plugging. It includes a module called subfs, which creates a dummy filesystem at the desired mountpoint. This module then performs mounting and unmounting operations before and after all filesystem operations. This way, the hardware can be removed at any time, without risk of data corruption.

The **OpenPOSIX** test suite has reached 1.0.0. This milestone contains tests for core POSIX conformance in the areas of signals, message queues, semaphores, timers and process scheduling. Although this is not a Linux-specific tool and Linux itself is not as concerned with POSIX conformance as other OSes, the OpenPOSIX test suite still is quite useful for areas where Linux does value conformance.

Martin Schlemmer found that **OSS** sound worked on his ICH5 (Intel I/O Controller Hub), simply by adding the ICH5 IDs to the list in the kernel sources. This could be useful for some systems, but as **Jeff Garzik** has pointed out, it does not work for all ICH5s out there.

QLogic has rewritten its **Fibre Channel** (FC) driver completely for its ISP21xx/ ISP22xx/ISP23xx chips and HBAs. The driver removes all support for the 2.4 kernel but adds significant performance enhancements. Their goal is to get the driver into the official 2.5 tree before the 2.6 time frame.

In preparation for **Serial ATA** (SATA), **Jeff Garzik** has created a driver to access ATA disks across the SCSI interface. He says the particular features of SCSI, as well as its advanced support for modern kernel features like SysFS, make it a very good host for SATA support. By going through the SCSI layer, he was able to rely on many such features that he otherwise would have had to code by hand.

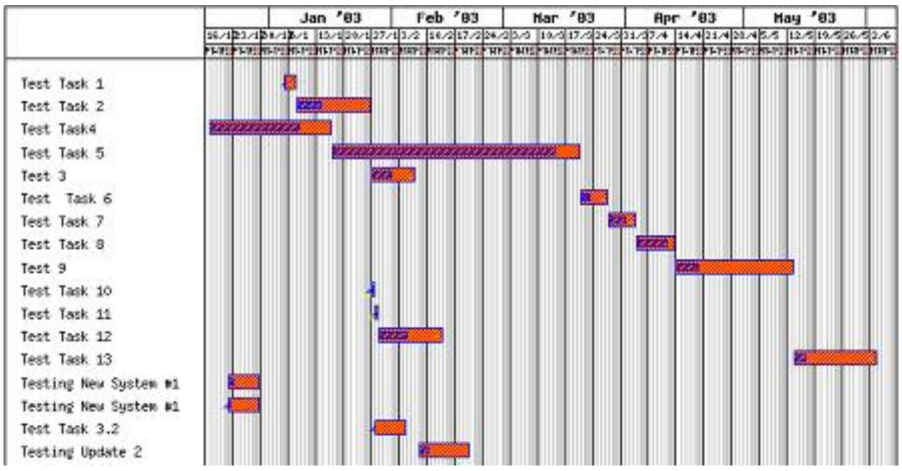
Ingo Molnar has announced the **Exec Shield** security feature, which provides protection against many (though not all) potential exploits. Stack, buffer and function pointer overflows all are guarded against, along with many other attacks. This is done without requiring any recompilation of user applications. Although not a complete solution, Exec Shield promises to be quite effective in conjunction with other security measures.

Development Management System in a Box: GForge

Don Marti

Issue #114, October 2003

GForge (gforge.org), the Web-based collaborative software development system, has released a 3.0 version. GForge combines features such as a bug tracking system, CVS version control, Web interface to CVS, and archived Mailman mailing lists. New features for 3.0 include a project management system with Gantt charts, along with internationalization and simpler installation. GForge is the actively maintained free software fork of the software that powers SourceForge.net. Requires: PostgreSQL, Apache, OpenSSL and PHP.



LJ Index—October 2003

- 1. Millions of requests for Python in May 2003: 11.9
- 2. Thousands of different servers making requests for Python in May 2003: 325
- 3. Dollars (equivalent to 188 million yen) the Japanese government will spend on a new IBM/Oki payroll system using Linux: 1,590,000
- 4. Number of Japanese government employees whose payrolls will be handled by the new payroll system: 800,000
- 5. Yearly costs to Japan in billions of dollars (350 billion yen) for its current payroll system: 2.96
- 6. Expected savings to the Japanese government in billions of dollars from implementing the new Linux-based system: 1.48
- 7. Dollars spent each year on proprietary software licenses by the South African government: 352
- 8. Years Linux has been in use by the Canadian National Railway: 10
- 9. Wireless LAN equipment sales in billions of dollars by 2006: 4
- 10. Thousands of technicians certified by the Linux Professional Institute: 27
- 11. Thousands of (Linux-based) TiVo customers: 750
- 12. Projected millions of households that will have TiVo-like PVRs (personal video recorders) by 2006: 30
- 13. Percentage of advertisers who say they will cut TV ad spending, based on projected PVR penetration: 76
- 14. Thousands of Linux-based HP workstations producing movies at DreamWorks: 1
- 15. Thousands of Linux-based processors in renderfarms at DreamWorks: 3
- 16. Millions of lines of code ported to Linux by Pixar: 300

- 17. Broadband penetration percentage in Korea: 70
- 18. Broadband penetration percentage in the US: 35.9
- 19. Apache market share percentage among top Web servers in July 2003: 63.72

- 1, 2: Guido van Rossum
- 3–6: Reuters
- 7: *InfoWorld*
- 8–10: CanadianBusiness.com, sourcing Forrester Research
- 11–13: *BusinessWeek*, sourcing Forrester Research
- 14–16: *eWeek*
- 17, 18: WebSiteOptimization.com
- 19: Netcraft (www.netcraft.com)

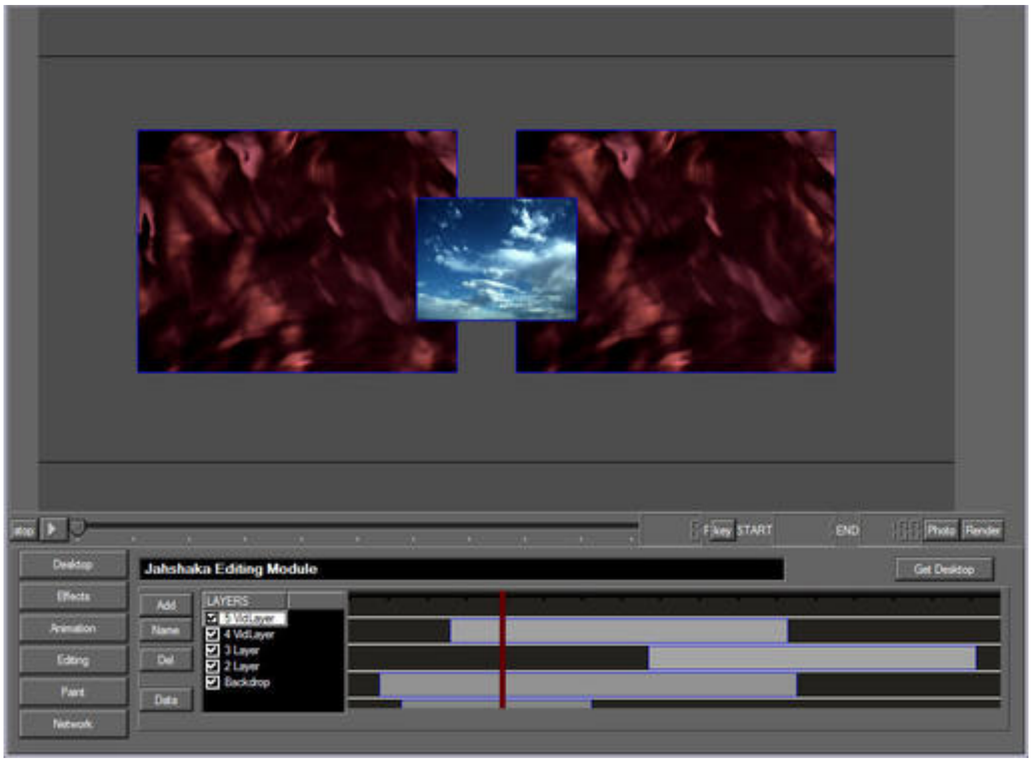
Jahshaka: Open-Source Real-Time Editing and Effects

Doc Searls

Issue #114, October 2003

Let's say you want to do real-time editing and effects—the kind of stuff you get with high-end Avid systems or applications like Adobe Premiere and Apple Final Cut Pro. You're going to spend thousands, and if you're really serious, the costs can run into six figures, easy.

Well, not anymore, thanks to Jahshaka, “The World's First Open-Source Real-Time Editing and Effects System”. The brainchild of Jah Shaka, a native of Kingston, Jamaica, and a music industry veteran, the project has (at the time of this writing) reached the alpha stage and runs on Linux, IRIX, Windows and (soon) OS X. Right now the modules include Effects, Animation, Editing, Paint, Text, Player and Compositor. Jahshaka is developed in the OpenML programming environment, which supports capturing, transporting, processing, displaying and synchronizing digital media. It's GPL'd too. See www.jahshaka.com and www.khronos.org/openml.



PhotoGen: shawley.myip.org/projects/photogen.php

David A. Bandel

Issue #114, October 2003

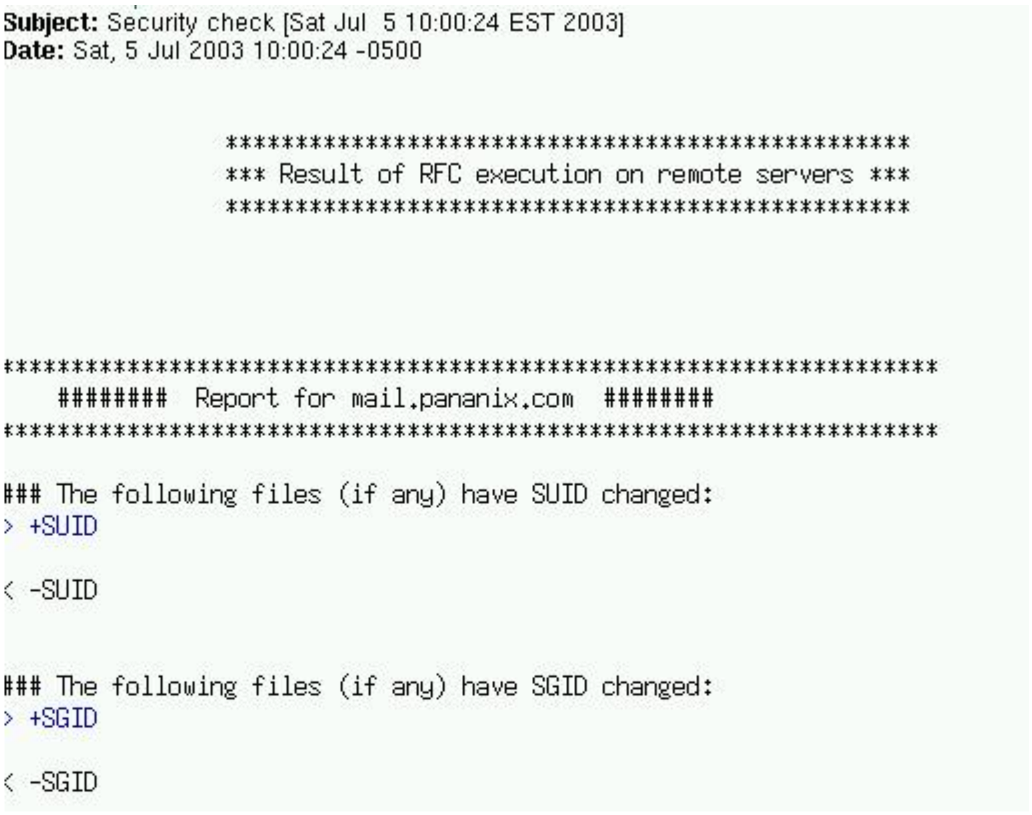
PhotoGen is another great, easy-to-use script to create a Web album of photos effortlessly. Scripts like this make it easy for anyone to create an index page of thumbnails and directly transfer the entire directory to a Web server. With a quick link to the directory, anyone can view your photos. Requires: bash and ImageMagick.

Remote Filesystem Checker: rfc.sourceforge.net

David A. Bandel

Issue #114, October 2003

The remote filesystem checker (RFC) allows you to run filesystem checks on a number of remote systems from one master node. RFC uses SSH to log in and check each system. This can be run nightly from cron and a report is sent to the administrator, which lets you see a number of systems on one report rather than getting one from each individual system. Requires: OpenSSH, BASH, cron, Perl and AIDE (optional).



```
Subject: Security check [Sat Jul  5 10:00:24 EST 2003]
Date: Sat, 5 Jul 2003 10:00:24 -0500


                ***************************************************
                *** Result of RFC execution on remote servers ***
                ***************************************************




**********************************************************************
    ########  Report for mail.pananix.com  ########
**********************************************************************

### The following files (if any) have SUID changed:
> +SUID

< -SUID


### The following files (if any) have SGID changed:
> +SGID

< -SGID
```

They Said It

Linux is now the new VHS.

—Jeff Gerhardt

Never doubt that a small group of thoughtful, committed citizens can change the world. Indeed, it's the only thing that ever has.

—Margaret Mead

In the age of the leak and the blog, of evidence extraction and link discovery, truths will either be out or be outed, later if not sooner.

—William Gibson

When it gets to the point where people are canceling publishing of books, it's very scary. It's a sort of censorship.

—Bill Pollock, No Starch Press, on the Digital Millennium Copyright Act, *New York Times*, July 10, 2003

The great anti-Linux mantra is gone. It has disappeared. Not one of the more than 60 responses mentioned a need for easier installation of the operating system. Kudos to everyone who helped to make that happen.

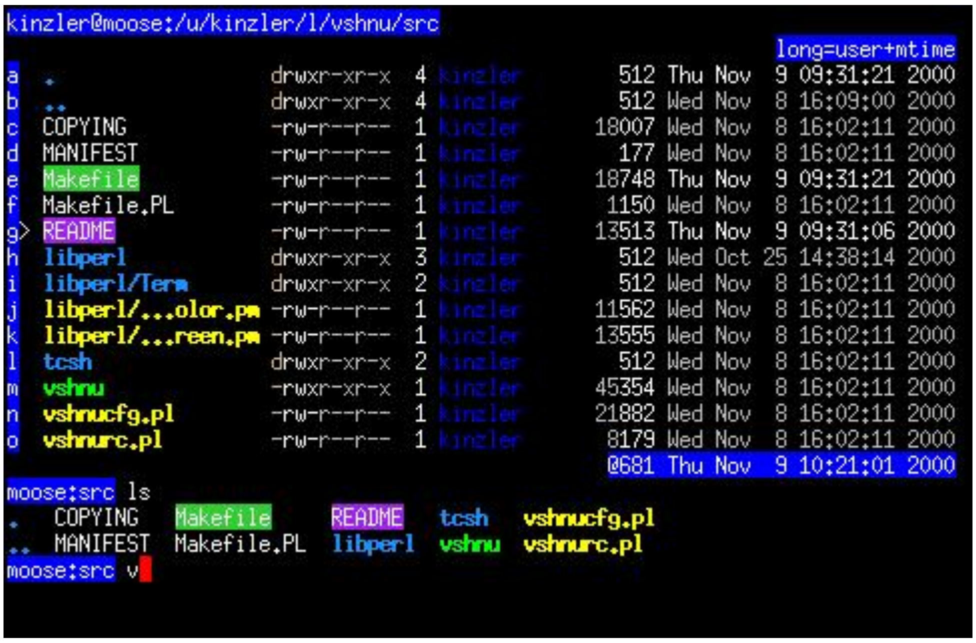
—Joe Barr, *NewsForge*

vshnu: www.cs.indiana.edu/~kinzler/vshnu

David A. Bandel

Issue #114, October 2003

A variation on a shell, this particular “shell” provides a graphical environment for a virtual terminal, while still allowing access to the underlying system. Properly setup, vshnu could serve as a batch menu file for new users. Written in Perl, all of Perl's power is available from vshnu. If you want a change from your standard CLI, this is worth a look, though it takes a little getting used to. Requires: Perl, Perl modules: Term::Screen, Term::ANSIColor, Term::ReadLine::Perl and a termcap file.



```
kinzler@moose:/u/kinzler/l/vshnu/src
                                                     long=user+mtime
a  .                  drwxr-xr-x  4 kinzler     512 Thu Nov  9 09:31:21 2000
b  ..                 drwxr-xr-x  4 kinzler     512 Wed Nov  8 16:09:00 2000
c  COPYING            -rw-r--r--  1 kinzler   18007 Wed Nov  8 16:02:11 2000
d  MANIFEST           -rw-r--r--  1 kinzler     177 Wed Nov  8 16:02:11 2000
e  Makefile           -rw-r--r--  1 kinzler   18748 Thu Nov  9 09:31:21 2000
f  Makefile.PL        -rw-r--r--  1 kinzler    1150 Wed Nov  8 16:02:11 2000
g> README             -rw-r--r--  1 kinzler   13513 Thu Nov  9 09:31:06 2000
h  libperl            drwxr-xr-x  3 kinzler     512 Wed Oct 25 14:38:14 2000
i  libperl/Term       drwxr-xr-x  2 kinzler     512 Wed Nov  8 16:02:11 2000
j  libperl/...olor.pm -rw-r--r--  1 kinzler   11562 Wed Nov  8 16:02:11 2000
k  libperl/...reen.pm -rw-r--r--  1 kinzler   13555 Wed Nov  8 16:02:11 2000
l  tcsh               drwxr-xr-x  2 kinzler     512 Wed Nov  8 16:02:11 2000
m  vshnu              -rwxr-xr-x  1 kinzler   45354 Wed Nov  8 16:02:11 2000
n  vshnucfg.pl        -rw-r--r--  1 kinzler   21882 Wed Nov  8 16:02:11 2000
o  vshnurc.pl         -rw-r--r--  1 kinzler    8179 Wed Nov  8 16:02:11 2000
                                               @681 Thu Nov  9 10:21:01 2000
moose:src ls
.   COPYING   Makefile     README   tcsh   vshnucfg.pl
..  MANIFEST  Makefile.PL  libperl  vshnu  vshnurc.pl
moose:src v
```

From the Editor

Don't Kill Peer-to-Peer, Make It Stronger

Don Marti

Issue #114, October 2003

Why you should block Kazaa and encourage better P2P.

At a recent demonstration of a network statistics tool, the presenter pointed out a bar on a chart and said that about half of the bandwidth used at the site was for a peer-to-peer (P2P) system called Kazaa.

P2P systems have enormous potential to free independent artists and Web sites from big ISP bills, sneak political information past censors and, perhaps most importantly for our readers, provide a challenging new platform to develop software ideas. But, Kazaa is proprietary, mostly unencrypted and, with a little work, can be filtered out. On page 56, Chris Lowth explains how.

Please do it. Read the article and filter your company's Kazaa users into oblivion. It's one of the projects you shouldn't have trouble convincing management to approve, because legal threats for hosting illegal copies of files on P2P systems are big news. With the techniques covered in Chris' article, you might be able to get a Linux firewall approved for a company that doesn't use Linux yet. There's always a first Linux project for every company, and this makes a good one.

However, there's another, more important reason to block Kazaa. The gap between what P2P needs to do in order to be a useful free and anonymous speech system and what Kazaa bothers to do, is shocking. If you can detect Kazaa traffic, repressive regimes certainly can. Help encourage the development of strong P2P by blocking inadequate P2P.

Ready to start working on a P2P system of your own? Brandon Wiley introduces a fundamental technique, distributed hash tables, on page 71.

And, because the theme of this issue is system administration, we cover two of the classic problems: how to design a backup plan for quick and painless restores and how to set up robust, sane file servers. Now that blank CDs are cheap, it makes sense to give each system its own custom recovery CD. Craig Swanson and Matt Lung cover Mondo for bare-metal restores on page 46. Erez Zadok lays out the problems with NFS and covers a flexible automount solution you can live with, on page 52.

After the movie industry, the next big leap for the Linux desktop is into electronic design automation (EDA). Not only are open-source EDA projects gaining features, the established EDA vendors are porting their proprietary tools to Linux as well. Michael Baxter explains how you can do hardware design almost at the speed of software with Xilinx chips and development tools on page 74.

Reuven Lerner's series on open-source content management systems for the Web continues with coverage of Bricolage on page 12. And, Eric Jeschke is back by popular demand with more GIMP techniques on page 80.

All you kernel hackers and scalability fans out there can check out page 18 where Paul McKenney explains the read-copy update (RCU) algorithm he developed at IBM and how he and other kernel hackers are making it work in the eagerly anticipated 2.6 kernel. Don't forget to download and test a 2.6 kernel with your own favorite apps and hardware. With the freedom to hack comes the responsibility to report bugs.

Don Marti is editor in chief of *Linux Journal*.

Best of Technical Support

Local-Only Domain Name?

When setting up a small home network (Red Hat 9), where your only connection to the Internet is through a dialed-PPP connection, what is the best practice for picking a domain name for the network so it won't clash with some valid domain name when you're connected to the Internet? Is there some kind of standard, internal-only domain name, similar to a hostname of localhost for 127.0.0.1?

—
Steve Cavender

SyntheSys@csi.com

You do not need any name; your Internet access will be fine without one. If you really want one, then register yourself a domain name.

—
Usman Ansari

uansari@yahoo.com

RFC 2606 reserves the .test top-level domain for testing; no .test domains will be assigned on the Internet. Or, if you don't want to pay to register a domain, and you have a friend who's into DNS, you could ask for a subdomain of one of his or her domains.

—
Don Marti

info@linuxjournal.com

Red Hat uses localdomain (your machine defaults to localhost.localdomain).

—
Marc Merlin

marc_bts@google.com

Act on a Change to a File

When using a cron job, I know a process is executed on a certain schedule. Is there any way to execute a process once there has been a change in a directory or file? If so, how would one do such a thing?

—
Iman Ahmadi

imanahmadi@hotmail.com

One way is to use Linux Directory Notification. You can find documentation and examples in the Documentation/dnotify.txt file under the kernel source.

—
Usman Ansari

uansari@yahoo.com

You can have your cron or at job check one of the timestamps of the file or directory. The easiest way do that in a shell script is to use GNU find's `"%T@"` or `"%C@"` format specifiers. This returns the modification date (%T) or the ctime (change status) date in seconds since the epoch. You can specify a number of other date formats if you like, but seconds since the epoch is handier for some forms of date arithmetic. A more-advanced approach would be to monitor the file or directory access with FAM (file alteration monitor). See oss.sgi.com/projects/fam/faq.html, but this is beyond a simple shell script.

—
Jim Dennis

jimd@starshine.org

I Have No init and I Must Boot

My Mandrake system shows the following error when I try starting up Linux:

```
   mount:error 22 mounting ext2 flags
   Kernel panic:no init found.
   Try passing init=option to kernel
```

I've successfully booted it three times, but thereafter it has stopped booting and keeps showing this message. What can I do?

—
Devi

deenadev_20@hotmail.com

A Linux kernel mounts the root filesystem and runs a program named init, usually found in the /sbin directory. You could read the kernel sources to find the other three or four places were Linux looks for a program named init. You also could pass the kernel a command-line or bootloader option `init=/bin/sh`, or some other full path to a program that the kernel would try to load. That's what the rest of this error message is trying to tell you. The question becomes: Why is your kernel unable to find the init program? The most likely cause is you are trying to mount something other than the root filesystem. If you tried to boot up and you passed the kernel a `root=/dev/hda6 ...` or some other device/partition path to a valid filesystem that didn't have /sbin/init on it, this is exactly the error message you'd receive. Actually, if it also didn't have a /dev/ directory on it, you'd get another error message: “Warning: Unable to open initial console.” That is not a fatal error, but perhaps you also were getting that error and not noticing.

If you had a completely bogus root= directive, if there wasn't a Linux-supported filesystem there, you'd get a different error, something like: “VFS Kernel panic. Unable to mount root.”

Other possibilities are that you had the system set up correctly but accidently removed or renamed /sbin/init or the /sbin directory, or your filesystem is so corrupt that /sbin/init's directory entry or inode is inaccessible.

In any event, I'd boot from a rescue disk, possibly a BBC (bootable business card), such as one you might download from www.lnx-bbc.org. Burn to a CD-ROM; a mini-CD or business card format usually are used for these, but a full-sized disk will work too. Boot from the BBC or Tom's Root/Boot (www.toms.net/rb), and run **fsck** on each of your filesystems. Then, look them over and figure out which one is really your rootfs. Next, try booting manually, passing the paramaters root= and init= using your bootloader (GRUB or LILO). Then try fixing the /boot/grub/menu.lst (grub.conf) or /etc/lilo.conf to list the correct kernel parameters properly (if that turns out to be the cause of the whole problem). While you have the system booted up from your BBC or other rescue media, it would be a good time to ask yourself: Is there any data I haven't backed up on these filesystems? Is now a good time to copy that data off to some other media? In the worst case you can re-install. There's no shame in saying: “My data is backed up; I don't understand all this fscking and those bootloader config files. Heck with it!” and then doing a re-install.

—
Jim Dennis

jimd@starshine.org

Minicom Is Not pppd

I am having trouble getting on-line. I am using Minicom and have set up PPP (using Red Hat 6.1). When I dial with Minicom, it appears that I connect, but then it asks for a login and password. I enter my user login and a bunch of charactors pop up, but then after a minute, it says “No carrier”. Any idea of what I am doing wrong?

—
Chris

calf@lanset.com

You aren't doing anything wrong; you're simply not using the right program for the task. Minicom is a terminal program; it cannot establish a PPP session and allow you to be a client of the remote network. You need a program like pppd, which should be part of your distribution. However, because your distribution is relatively outdated, you may have problems connecting to your ISP, depending on the version of pppd included. If so, you should install a newer version of pppd or upgrade to a newer distribution, depending on your skill and comfort level.

—
Chad Robinson

crobinson@rfgonline.com

Try the graphical kpppd or something similar. You also may want to update your distribution. Currenly Red Hat 9.0 is out, and you will be pleasantly surprised by it.

—
Usman Ansari

uansari@yahoo.com

Connecting to a Company VPN

The company for which I work has a VPN that uses Microsoft's VPN server. Is it possible to connect to this using Linux (Red Hat 9)?

—
Dean Siewert

dsiewert@execpc.com

There is a Linux point-to-point tunneling protocol (PPTP) client compatible with Microsoft's VPN servers. Visit pptpclient.sourceforge.net.

—
Chad Robinson

crobinson@rfgonline.com

Be sure to find out which protocol they are using. The standard way of creating a VPN with Microsoft products was the built-in PPTP. Recently, they have adopted IPSec and L2TP.

—
Mario Bittencourt

mneto@argo.com.br

A PPTP server for Linux is the PoPToP package, www.poptop.org.

—
Jim Dennis

jimd@starshine.org

Re: August 2003, Best of Tech, “Dual-Boot System Skips LILO Menu”

Both answers as printed will work, but both assume the existence or availability of a rescue floppy. If there isn't one, a fix can be made with the first Red Hat install CD: 1) Boot from Red Hat 9 CD #1. 2) Type the boot option `linux rescue`. 3) Choose whatever language and keyboard options are relevant. 4) At the Rescue prompt, choose Skip. 5) At the shell prompt, type `cd /mnt`. 6) Type `mkdir sysimage`. This folder could be named anything. However, Red Hat included a folder by this name for this reason back in Red Hat 7.3, so I'm including it here for mnemonic purposes. 7) Type `mount /dev/`*`WHATEVER`* `sysimage/`. Mount your / partition here. 8) Type `chroot sysimage`. 9) Mount /usr and /boot partitions, if necessary. 10) Type `grub` or `/sbin/grub`. 11) Type `root (hd0,0)`. This tells GRUB where to find its second-stage loader and config files—in this example, the first partition of the first hard disk. 12) Type `setup (hd0)`. GRUB is placed in the MBR of the first HS. 13) Type `quit`. 14) Type `exit` until you are logged out of all shells and Red Hat reboots. `info grub` has a lot of good info that would flesh out this bare-bones tutorial.

—
Daniel Callahan

proteus@eclectic-cheval.net

On the Web

Toward an Open-Source Government

Heather Mead

Issue #114, October 2003

If we can't always have the leaders we want, can we at least have some open-source technology for our tax dollars?

Many people in the Open Source community have long believed that open-source software and the government are a good match. As any state-level politician can verify, US state governments are faced with the worst budget conditions in at least 50 years. In addition to having to gut and cut programs and services possibly needed by more people than ever, many schools and state-funded universities are expected to provide more technology for more students with the same or less money. In light of these conditions, some proponents of open source believe now is a perfect opportunity for states to save money by moving legacy systems to and creating new systems that run on open-source software. It's no longer a maybe, someday thing, if it's a small and adventuresome state agency.

Getting governments to make these switches, however, is where logic and progress can hit a wall of regulation, procedure and red tape. State and local governments seem to be the best place to start the move for change, so if we're going to give it a shot, we better know what the current procedures are. To that end, Tom Adelstein has been writing a weekly series for the *Linux Journal* Web site that covers the current state of open-source in government. “Linux Access in State and Local Government, Part I”, /article/6927, reviews specific states' endeavors to legislate consideration of open-source software and details some of the detours these endeavors encounter. In Parts II and III, /article/6952 and /article/6970, Tom covers particular state agencies in Texas and Oregon and explains where they were successful in using open source and where they were not. Finally, in Part IV, /article/6990, Tom walks us through the process of getting IT legislation through state government.

In a similar vein, Joe Barr's article, “Austin, Texas to Begin Linux Pilot Project”, /article/6974, describes how the IT department for the city of Austin is launching a city-wide program to use Linux desktops, servers and thin clients. The city's new CIO is “especially interested in seeing what sort of performance and savings he can get out of running office applications on a server with terminals as the desktop”. If cities and states can get out from under some of the proprietary licensing fees and TCO, perhaps more money will be available for all those national defense and antiterrorism programs that states are supposed to enact without extra national funding.

Moving on to something a little lighter, here's your project for the month. Use Jeffrey Taylor's “Peering Over the Firewall” how-to (/article/6985), Snort, a custom cable and an Ethernet hub to build your own low-cost intrusion detection system. By the end of the project, the Linux box will be the firewall and router in your home network, and you'll still be able to peer around the router and see all incoming packets.

If you have an open-source-in-government story or a do-it-yourself project you'd like to share, send your proposal to info@linuxjournal.com. Be sure to check the *Linux Journal* Web site often; new articles are added daily.

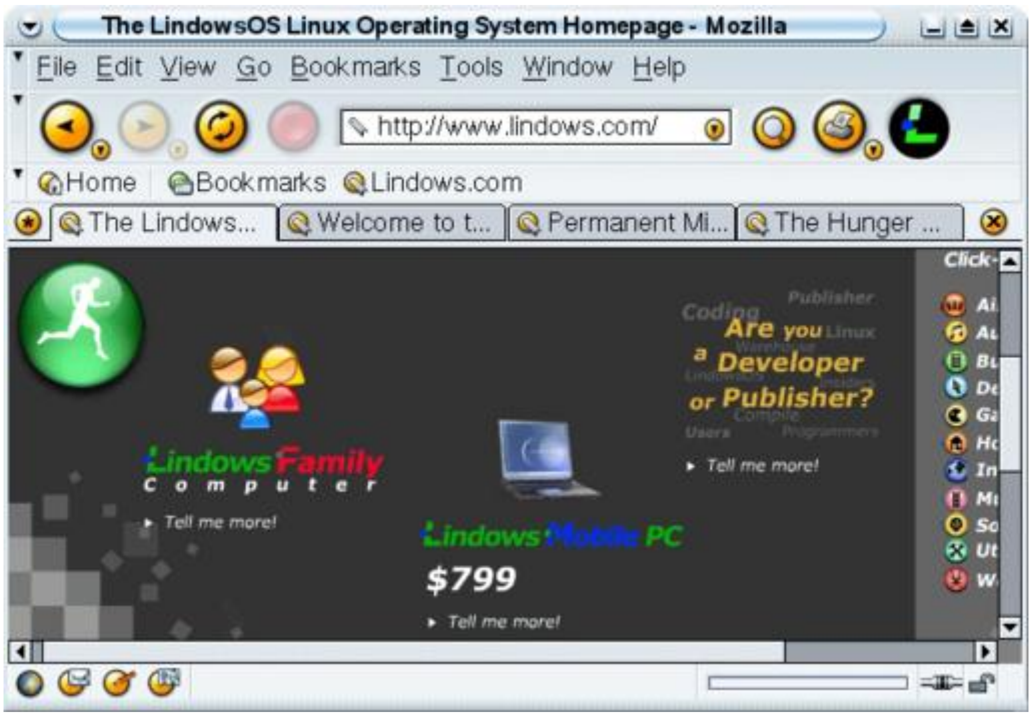
Heather Mead is senior editor of *Linux Journal*.

New Products

Lindows 4 and LindowsCD

LindowsCD is a version of Lindows 4 designed to run from a CD-ROM without installation or setup. LindowsCD includes many features of Lindows 4, the newest full version of Lindows OS. Features found in both Lindows 4 and LindowsCD include hardware detection, Bayesian spam filtering, a new IM program and multimedia support for MP3, Real Audio, Real Video and Macromedia Flash. LindowsCD is designed for educational and training environments and Linux testing. When LindowsCD is removed, the computer reverts to its original state.

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


The LindowsOS Linux Operating System Homepage - Mozilla
File Edit View Go Bookmarks Tools Window Help
http://www.lindows.com/
Home Bookmarks Lindows.com
The Lindows... Welcome to t... Permanent Mi... The Hunger ...
LindowsFamily Computer
Tell me more!
Are you a Developer or Publisher?
Tell me more!
LindowsMobile PC
$799
Tell me more!

PCS-620 Wireless SBC

Octagon Systems released a compact SBC that combines wireless, Ethernet, serial and industrial I/O and comes with Linux onboard. The card supports up to 512MB of SDRAM and 2GB of Flash over a CompactFlash socket. The PCS-620 incorporates a low power 300MHz Pentium processor and supports dual 10/100BaseT Ethernet ports; four serial ports; RS-232/444/485; a CardBus port for wireless, GPS or other devices; dual USB ports, dual AC97-compatible audio, CRT and flat-panel video to 1280 × 1024; parallel, floppy and EIDE ports; and an ambient temperature sensor. The CompactFlash requires no drivers and is compatible with most OSes.

Octagon Systems Corp., 6510 W 91st Avenue, Westminster, Colorado 80031, 303-430-1500, www.octagonsystems.com.

SpamPlug

SpamPlug, from Mail-Filters.com, Inc., is an outbound antispam product. Based on the SpamCure filter for inbound mail, SpamPlug monitors e-mail traffic on its way to the Internet; spam is rejected or held for review while other messages are delivered immediately. SpamPlug uses a constantly updated spam signature database to detect spam, improving chances of catching actual spam messages and lowering the potential for false positives. SpamPlug installs quickly on a dedicated server or can co-reside on the e-mail server.

Mail-Filters.com, Inc., 205 De Anza Boulevard #200, San Mateo, California 94402, 650-655-7700, www.mail-filters.com.

Meteor Laptop

Based on the Sharp Actius MM10, EmperorLinux's Meteor is a two-pound, one-half-inch thick laptop with Linux pre-installed. The Meteor MM10 comes with full Linux hardware support for X, sound, USB, PCMCIA, Wi-Fi, networking and more. The laptop features a 10.4" Active XGA TFT that runs X at 1024 × 768 × 24bpp with SMI Lynx graphics. It also has a 1,000MHz Crusoe processor, 512KB of L2 cache, 256MB of RAM, a fixed 15GB hard drive, one type II PCMCIA slot and two USB ports. The USB cradle allows you to treat the MM10 as a USB hard drive while it is off, so you can mount the MM10's drive from other Linux boxes and sync. The Meteor can be installed with the EmperorLinux, Slackware or

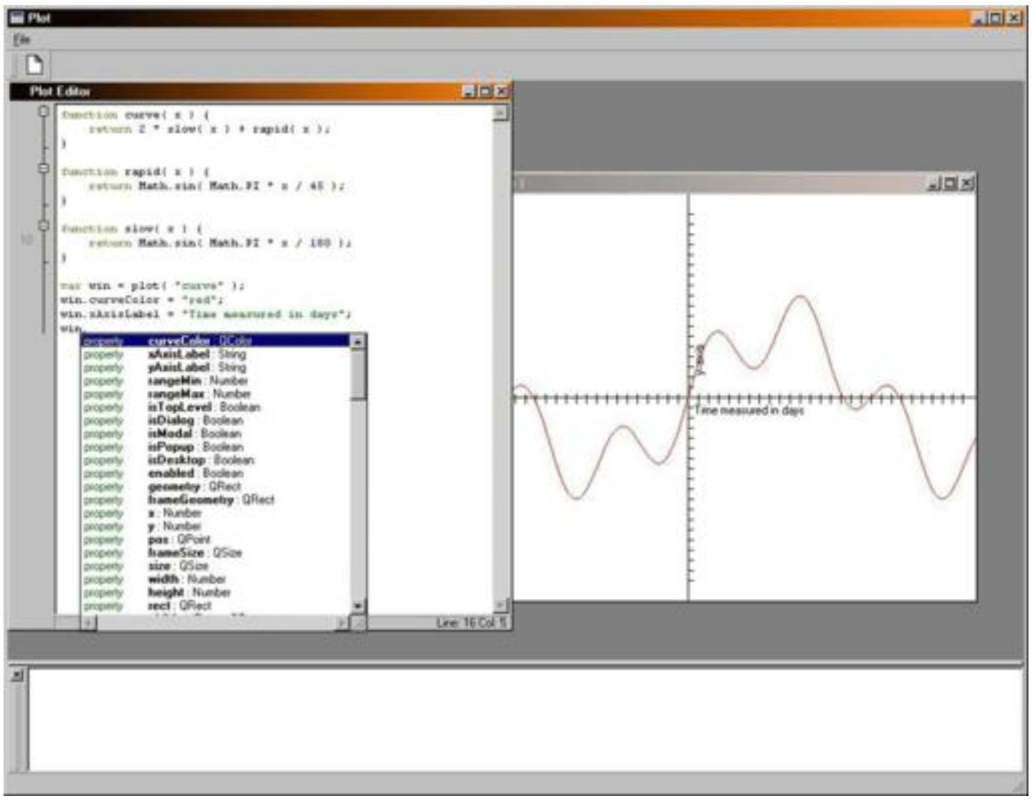
Mandrake distribution, with an option for a dual boot with Microsoft Windows XP. It also comes with a care package, one year of support and various options.

EmperorLinux, 900 Circle 75 Parkway, Suite 1380, Atlanta, Georgia 30339, 770-612-1205, www.emperorlinux.com.

QSA

Trolltech's new QSA (Qt Script for Applications) toolkit enables Qt applications to be scriptable. Adding a few lines of code to a Qt application enables it for scripting and extending. QSA has four components: QSA SDK, a kit for making applications scriptable; Qt Script, a multiplatform interpreted scripting language; QSA Workbench, a scripting environment; and Input Dialog Framework, a GUI API for writing dialogs. With these components, users can prototype changes in already-compiled applications, write test scripts, customize applications, provide script-based fixes for already-released applications and more.

Trolltech Inc., 1860 Embarcadero Road, Suite 100, Palo Alto, California 94303, 650-813-1676, www.trolltech.com.



```
function curve( x ) {
    return 2 * slow( x ) + rapid( x );
}

function rapid( x ) {
    return Math.sin( Math.PI * x / 45 );
}

function slow( x ) {
    return Math.sin( Math.PI * x / 180 );
}

var win = plot( "curve" );
win.curveColor = "red";
win.xAxisLabel = "Time measured in days";
win.
```

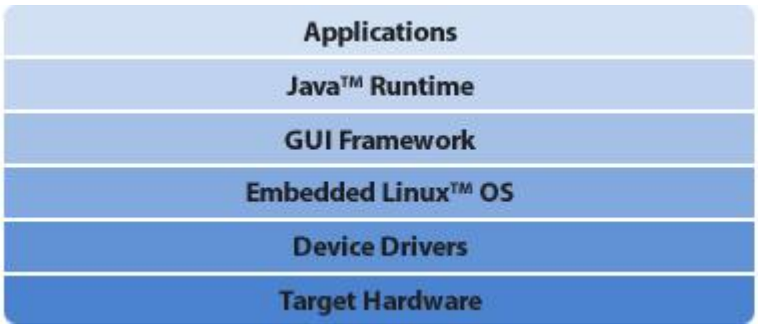
```
property    curveColor : QColor
property    xAxisLabel : String
property    yAxisLabel : String
property    rangeMin : Number
property    rangeMax : Number
property    isTopLevel : Boolean
property    isDialog : Boolean
property    isModal : Boolean
property    isPopup : Boolean
property    isDesktop : Boolean
property    enabled : Boolean
property    geometry : QRect
property    frameGeometry : QRect
property    x : Number
property    y : Number
property    pos : QPoint
property    frameSize : QSize
property    size : QSize
property    width : Number
property    height : Number
property    rect : QRect
```

OpenPDA Development Studio

OpenPDA Development Studio for the AMD Alchemy Solutions DBAu1100 is a joint effort by Metrowerks and AMD to offer a platform for building low power consumption, high-performance personal connectivity devices. OpenPDA

Development Studio combines software, the Alchemy processor, memory capabilities and peripherals on a single board. The MIPS-compatible processor operates at speeds up to 500MHz using .5 watt of power. OpenPDA includes hardware and software for gaming, multimedia player, browsing and Java applications, as well as a PIM suite. The platform includes an embedded Linux kernel, Qtopia, the Jeode JVM and Opera.

Metrowerks Corporation, 9801 Metric Boulevard, Austin, Texas 78758, 512-996-5300, sales@metrowerks.com, www.metrowerks.com.



Magnia SG30 Wireless Server

Toshiba's SG30 Wireless Server enables collaboration by providing fast and easy file sharing, printer networking and Internet access. Designed for use in small- to medium-sized businesses or remote offices, the Magnia is about the size of a laptop and enables data sharing in and away from the office. In addition to Internet and printer gateway Web caching, content filtering, backup capabilities, e-mail services and remote administration, the Magnia also supports 802.11b IEE Wi-Fi. It comes with three PCMCIA slots for wireless LAN PC card support, a built-in wireless access point, a Celeron processor with 256KB L2 cache, 256MB of memory, 128-bit Wi-Fi encryption, a configurable firewall and VPN, up to 160GB of storage capacity and two 2.5" EIDE drive bays.

Toshiba America Information Systems, Inc., 9740 Irvine Boulevard, Irvine, California 92618, www.toshiba.com.