Linux Journal Issue #87/July 2001

Features

Focus: Program Development *by Don Marti*
Debugging Memory on Linux *by Petr Sorfa*
Sorfa provides some examples of multiple debugging methods.
CVS: An Introduction *by Ralph Krause*
Krause explains the workings and uses of this version control system.
Create User Interfaces with Glade *by Mitch Chapman*
Discover the joys of creating GUI apps with Glade and Python—Chapman shows us how.

Indepth

Automating Firewall Log Scanning *by Leo Liberti*
Liberti gives some clues for increasing security and saving time by automating log scanning.

Toolbox

At the Forge Custom JSP Actions *by Reuven M. Lerner*
Cooking with Linux Programming Silence OUT! *by Marcel Gagné*
Paranoid Penguin Intrusion Detection for the Masses *by Mick Bauer*
GFX Linux at NAB *by Robin Rowe*

Focus: Program Development

Don Marti

Issue #87, July 2001

Have fun with these development tools, but don't forget to take a break from coding and read something.

Read more. I know it sounds like something you might see on a bookmobile poster, but seriously, read more. There are so many tools for writing stuff, and so many places to distribute what we write, that the appeal of writing is getting dangerous. Meanwhile, reading isn't getting any easier.

The first high-level advice every programmer gets is “Don't write what you can reuse.” Good, but you need to read in order to learn what to reuse. Effective code reuse isn't just a matter of “I need SSL, so I'll do a search on +ssl +library.” Congratulations, you found OpenSSL, but how did you know SSL would help you? Because, not having lived in a cave for the last ten years, you read about it somewhere. What else—maybe something not as famous as SSL—could you use? If you don't read, you'll never learn and just end up writing it yourself.

Brian Behlendorf once wrote, “The world needs less software.” That doesn't mean we shouldn't write software if we have to, just that we shouldn't write it for the wrong reasons. At first it's more fun to write your own thing than to grovel through and understand somebody else's, but someday that nifty new thing will be the stuff you wrote a long time ago and are sick of maintaining. Learn to reuse and you're learning to make other people do work for you.

A contributor to one project's mailing list asked the question, “Should I get in the credits list for a patch that only removes code?” It was a GUI project, and the patch made a dialog box use some default layout settings instead of a tightly-controlled but broken layout. Of course you should get in the credits list for removing code. You made the program better and smaller at the same time. And you provided an example for others. Removing code can take more understanding and skill than adding it, and people who can do it successfully should be at the top of the credits list.

Remember XCoffee? It was a great idea—a special-purpose server application that just serves up coffee pictures over the Net and a special-purpose client application that just displayed coffee pictures. Pretty advanced for its time, but just coffee pictures? And just one picture per instance of the client? The original XCoffee got replaced with a web site—no special client software, it works on clients without X, and all the hard-to-get-right network stuff is handled by the web server. If you knew of a general solution that let you slaughter your offspring, would you do it?

Speaking of reading, what's in this issue? Until something better comes along, if you want to do source code control, or browse many projects' development code that hasn't been officially released, you need to know CVS. Ralph Krause gives you the basics on page PAGE.

Memory management errors cause everything from the first segmentation fault in your first C program to security flaws that destroy entire companies. Improve your memory-fu by reading up on helpful free tools, some of which are built into the GNU C library. Petr Sorfa explains memory management and memory troubleshooting on page PAGE.

If you're a crusty old UNIX bastard who's been with us since issue one, you're probably getting set to scoff when we mention anything having to do with integrated development environments. So you go scoff over there while I tell these nice people about KDevelop and Glade.

On page PAGE Mitch Chapman introduces Glade, a fun tool that lets you break out GUI design from coding. Point and click to design the GTK interface, and write the code separately. Best of all, he's using Python for the examples, which fits nicely into the exciting new “use an object-oriented scripting language for the stuff that doesn't have to be fast” coding philosophy.

Not to exclude the KDE side of the desktop wars, Petr Sorfa explains KDevelop on page PAGE in the Product Review section. Naturally, as a KDE application, it's focused on C++ and Qt, and for a C++ programmer, to try the Qt GUI toolkit is to love it. Qt has an aesthetic cleanliness about it that C++ people seem to find difficult to express in words, but that's okay. KDevelop integrates with CVS—very neato—and with Qt Designer, which as you might guess is a user-interface design tool for Qt (look for an article on Qt in our next issue).

Have fun with these development tools, but don't forget to take a break from coding and read something.

—Don Marti, Technical Editor

Debugging Memory on Linux

Petr Sorfa

Issue #87, July 2001

Petr explains how programmers can prevent nasty program memory bugs.

All programs use memory, even ones that do nothing. Memory misuse results in a good portion of fatal program errors, such as program termination and unexpected behavior.

Memory is a device for handling information. Program memory is usually associated with the amount of physical memory a computer has but can also reside on secondary storage, such as disk drives, when not in use. Memory for users is managed by two devices: the kernel itself and the actual program using calls to memory functions such as malloc().

Kernel Memory

The operating system kernel manages all the memory requirements for a particular program, or instances of a program (because operating systems can execute several instances of a program simultaneously). When a user executes a program, the kernel allocates an area of memory for the program. This program then manages the area of memory by splitting it into several areas:

- Text—where only the read-only parts of the program are stored. This is usually the actual instruction code of the program. Several instances of the same program can share this area of memory.
- Static Data—the area where preknown memory is allocated. This is generally for global variables and static C++ class members. The operating system allocates a copy of this memory area for each instance of the program.
- Memory Arena (also known as break space)--the area where dynamic runtime memory is stored. The memory arena consists of the heap and unused memory. The heap is where all user-allocated memory is located.

The heap grows up from a lower memory address to a higher memory address.

- Stack—whenever a program makes a function call, the current function's state needs to be saved onto the stack. The stack grows down from a higher memory address to a lower memory address. A unique memory arena and stack exists for each instance of the program.

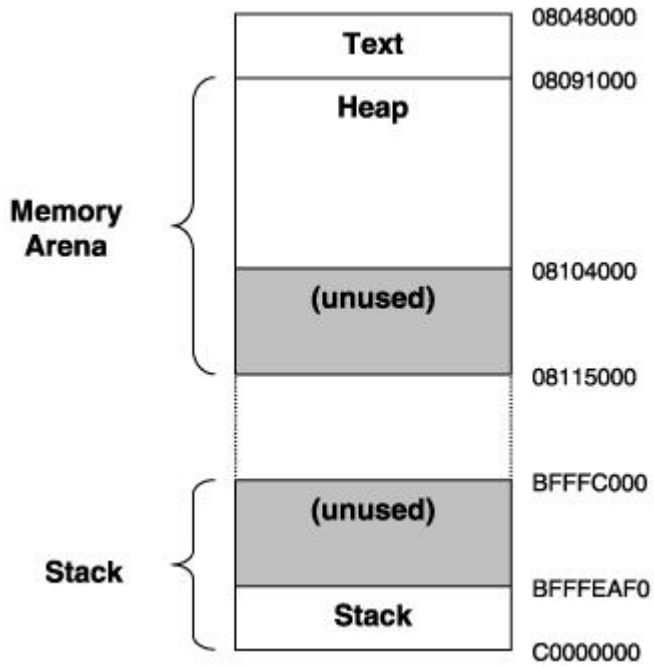


Figure 1. Memory Associated with an Instance of a Program

User Memory

User-allocatable memory is located in the heap in the memory arena. The memory arena is managed by the routines malloc(), realloc(), free() and calloc(). They are part of libc. However, it is possible to substitute these functions with another implementation that may provide better performance for a particular use. See sidebar for a list of alternate memory functions.

Alternate Memory Functions

On Linux systems, programs expand the size of the memory arena in precalculated increments, usually one memory page in size or aligned with a boundary. Once the heap requires more than what is available in the memory arena, the memory routines call the brk() system call that requests additional memory from the kernel. The actual increment size can be set by the sbrk() call.

To view the current stack and memory arena of any process, look at the contents of /proc/<pid>/maps for a particular process, where pid is the process id (see Listing 1).

Listing 1. Output from /proc/<pid>/maps

Structure

Each time new memory is allocated with malloc(), a little more memory is obtained than requested. The memory routines use this extra memory for maintenance. To obtain the real amount of memory allocated for user manipulation, use the function call malloc_usable_space(). The real memory chunk is usually eight bytes larger.

The structure of a memory chunk has the size of the chunk prepended and added to the end of the chunk (see Figure 2). The size value also has a bit flag that indicates whether the memory management system maintains the memory chunk immediately before the current one.

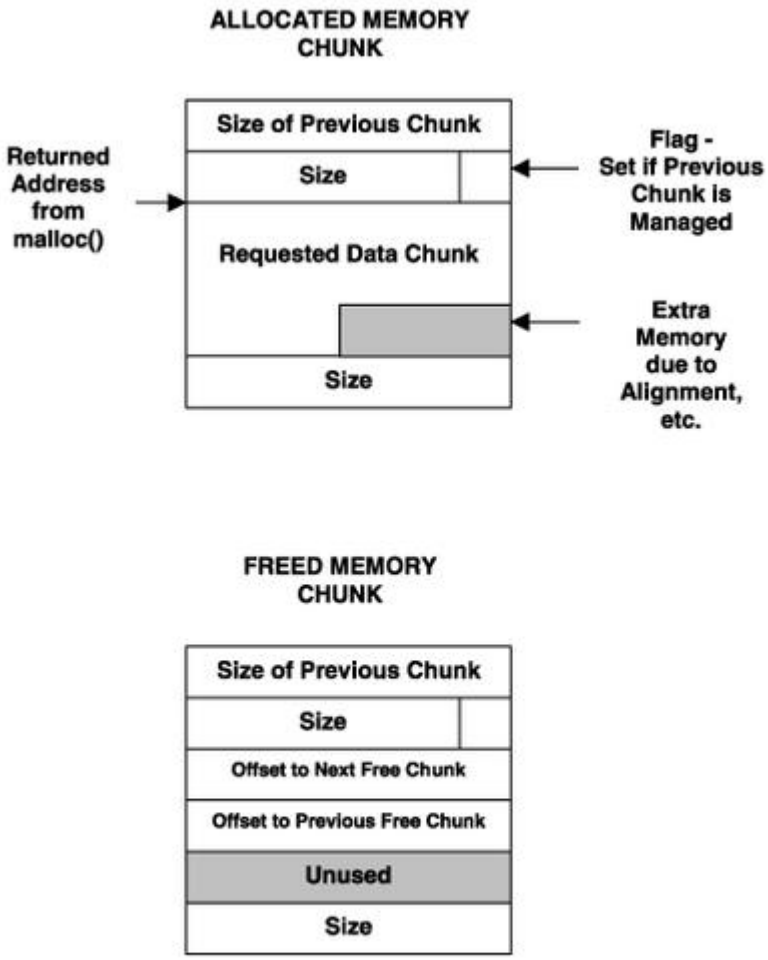


Figure 2. The Memory Chunk Structure

The memory routines in GNU libc use bins to store memory chunks of similar size to assist in improving performance and preventing fragmented memory areas, where you have unused memory gaps throughout the memory arena. These memory routines are also threadsafe. Though these routines are quick and stable, there may be areas of possible improvement, such as speed and memory coverage.

Debugging

Memory can cause bugs and usually unwanted memory behavior. One way is by the usage of freed memory, which is the usage of a memory chunk that the program has already freed. Although this will not necessarily cause problems immediately, something will go wrong once a new memory allocation takes over that same area of memory. As a result, the same memory area is used for two different purposes, which causes unexpected values that may lead to a program core dump if the memory area contains pointer values or offsets.

Another problem is trampling over the preamble to a memory chunk. If the program overwrites the preamble to a memory chunk, the memory management system will possibly fail or act unexpectedly when encountering the corrupted memory chunk.

Sometimes trampling occurs over an adjacent memory chunk, and this might corrupt data. The user might only pick up this kind of error later during program execution with odd values and program behavior.

Similarly, if the management information of a freed memory chunk is wrecked by trampling or unwarranted use, it is highly likely that the memory management system will cause an error.

Usage of the unallocated space in the memory arena could also have an effect. It may be possible to use the memory outside of the heap, which is still within the memory arena. This generally will not cause errors until newly allocated memory uses some of this space. This error could be very difficult to detect because the subsequent memory actions could keep within the heap space.

The most obvious and immediate error is when a program attempts to use memory outside of the memory arena and the program memory scope. This results in a SIGSEGV (segmentation violation fault), and the program will automatically dump core.

The most damaging and trickiest-to-debug memory error is when the stack of the program is corrupted. The program stores local variables, parameters and registers from previous frames and, most importantly, the return address in the stack. So if the stack becomes corrupted, the program may become

impossible to debug with a conventional debugger, as the stack frames themselves are rendered useless. Debugging stack memory problems is limited to a few open-source (e.g., libsafe) and proprietary memory debuggers because program execution needs to be altered or enhanced to detect stack memory violations.

There are several ways of attempting to catch and find memory misuses. Unfortunately, some have side effects, such as slower program execution speed and more memory usage, and consequently, they may be unusable in memory-intensive programs.

The buggy program examples used with the following memory debuggers can be seen in Listings 2, 3 and 4.

Listing 2. mytest00.c Example Program

Listing 3. mytest01.c Example Program

Listing 4. mytest02.c Example Program

By default there is an environment variable, MALLOC_CHECK_, that can be set to enable rudimentary debugging with the default malloc. MALLOC_CHECK_ can be set to one, in order to provide some error reporting, or set to two to abort the program whenever any malloc error occurs. The output can be cryptic because the debug mode reports problem areas as addresses rather than readable symbols. As a result, it is a good idea to have a debugger on hand to determine where in the program these errors are occurring. The following is an example using default memory debugging:

```
<home>$ MALLOC_CHECK_=1 ./mytest00
malloc: using debugging hooks
hello Linux users
free(): invalid pointer 0x80496d0
hello again
free(): invalid pointer 0x80496d0
realloc(): invalid pointer 0x80496d0
malloc: top chunk is corrupt
hello there
```

The output indicates the problem in mytest00.c, line 8 (Listing 2), where the strcpy() function overflows and corrupts the memory chunk pointed to by msg. The subsequent debugging messages are because of this corruption.

There are several excellent open-source memory tools available (see sidebar for a list). Each implementation differs in memory bug coverage, output and interaction.

Open-Source Memory Tools

Electric Fence is one tool that is simple to use. The library performs several memory checks and when encountering an error, stops the program. This usually results in a core dump, which the user then can investigate with a debugger. Electric Fence is most useful when employed within a debugger, such as the GNU debugger (GDB). When Electric Fence stops the program, GDB regains control at the exact location in the program where the error occurred (see Listing 5).

Listing 5. Memory Debugging with Electric Fence within GDB

This example output shows the test built with the Electric Fence library executing under GDB. The very first violation at mytest00.c line 8 results in a SIGSEGV. When examining the stack trace provided by GDB, the user can identify the problem location.

libsafe is used to check a number of possible stack frame boundary violations limited to a few C functions (strcpy, strcat, getwd, gets, scanf, vscanf, fscanf, realpath, sprintf and vsprintf).

The libsafe example output is terse. As soon as a stack error occurs, libsafe displays an error and terminates the program. However, libsafe sends the details of the actual error to various e-mail recipients. Granted, this is a convoluted way of reporting the error, but users primarily use libsafe to detect attempted security breaches that exploit buffer overflow. With a bit of editing, a developer can enhance the libsafe code to report messages that are more informative. Another option is to execute the program in GDB and set a breakpoint on _libsafe_die(), which is hit as soon as a stack violation is detected by libsafe. In the following example libsafe detects stack trampling caused by strcpy() in line 8 of mytest01.c (Listing 3):

```
<home>$ LD_PRELOAD=/lib/libsafe.so.1.3  ./mytest01
Detected an attempt to write across stack boundary.
Terminating mytest01.
Null message body; hope that's ok
# Email is the sent with the following subject header
libsafe violation for /tmp/mytest01, pid=27265;
overflow caused by strcpy()
```

debauch limits its output to contain addresses instead of symbols, which makes it necessary to be used with a debugger. **debauch** has special capabilities that users can activate specifically for GDB use. These capabilities allow better tracking of memory allocation and deallocation calls. **debauch** is thorough and detects and recovers from many of the memory errors (see Listing 6).

Listing 6. Memory Debugging with debauch

memprof's main feature is the GUI interface, which makes it easy to understand and to see where memory leaks occur. It has fairly powerful capabilities due to the fact that it utilizes functions that GDB uses to control processes via the binary file descriptor (BFD) library. Figure 3 shows that memprof has detected the leak in the function alloc_two() in mytest02.c.

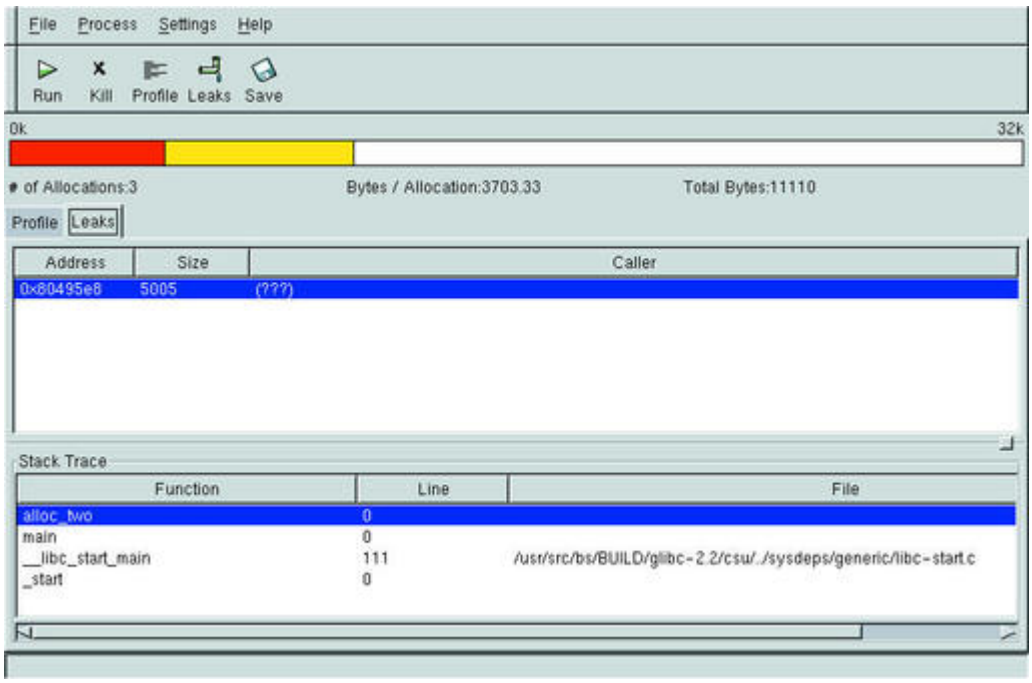


Figure 3. Memory Debugging with memprof

Apart from open-source memory tools, several proprietary tools are available that provide graphical user interfaces and more thorough checks than open-source versions (see sidebar for a list of proprietary memory tools).

Proprietary Memory Tools

Possibly, the last option is to write your own memory handling functions. This might be useful in becoming familiar with memory management or providing performance enhancement due to your particular needs, such a quick allocation and deallocation of large memory areas.

Debugging memory problems is important, for not only program stability, but security as well. There are several memory debuggers available for Linux, each with their own particular set of capabilities and usage criteria. The best approach is to test a program with more than one of these memory debuggers with a debugger such as GDB, as the combined power may detect a wider range of memory problems.

Petr Sorfa (petrs@sco.com) is member of Santa Cruz Operation's Development Systems Group where is the maintainer of the cscope and Sar3D open-source projects. He has a BSc from the University of Cape Town and a BSC Honours from Rhodes University. His interests include open-source projects, computer graphics, development systems and sequential art (comics).

Archive Index Issue Table of Contents

Advanced search

CVS: An Introduction

Ralph Krause

Issue #87, July 2001

Take advantage of the ability to track file versions, collaborate on projects and get back yesterday's work.

If you have downloaded software from the Internet, especially from SourceForge, you probably noticed the letters CVS. CVS stands for Concurrent Versions System and is a tool that allows developers to keep track of their projects. It also allows developers to collaborate on projects.

While CVS may be used on large projects with many developers over a network, this article focuses on its usefulness for individuals on local systems. A common occurrence with CVS might begin with making changes to a script or configuration file and then moving on to other tasks. After some time has passed, you find your changes aren't working, and you don't have a backup of the original file, and you can't quite remember all the changes you made. CVS can help prevent this situation because it keeps track of changes made to files and allows you to revert to working versions of them.

A Brief Overview of CVS

Files under the control of CVS are stored in a special directory called a repository, and each file has a revision number maintained by CVS. To make a change to a file, you first must get a copy of it from the repository. You can get a copy of the latest revision of a file or any earlier revision stored in CVS. When you are through working with the file, you put it back into the repository, and its revision number increases incrementally. Each time you commit a file to the repository you can supply a log message that helps keep track of which changes were made to the file over time.

CVS differs from other version control systems in that it doesn't lock files; different developers can check out a file and work on it at the same time. CVS makes sure one developer's edits don't conflict with edits made by another

developer when the file is put back into the repository. If conflicts are found, CVS places markers in the second developer's copy of the file, allowing him or her to find and resolve the conflicts. Once the conflicts are resolved, the developer then commits the file to the repository.

Installing CVS

There is nothing tricky involved in installing CVS. You can either download and compile the source or install an RPM package or its equivalent.

Once CVS is installed you will have to decide where to locate the repository. It should be in a partition that has a good amount of free space and one to which you have write permission. Once you have decided on a location for the repository, you have to create it and populate it with CVS administration files. This is done using the CVS **init** command. If you want your repository to be in /usr/local/cvsstuff, you would execute the CVS command shown below:

```
cvs -d /usr/local/cvsstuff init
```

The CVSROOT environment variable, or -d switch, tells CVS commands which repository to act upon. The CVSROOT variable can be set by adding the following line to your .bash_profile:

```
export CVSROOT=/usr/local/cvsstuff
```

Populating the Repository

To put existing projects into the repository, use the **import** command. For example, say you have the directory structure shown below and will add other client directories in the future.

```
html_projects/
         client1/
            images/
         client2/
            images/
```

To place html_projects and everything under it into the repository, you would use the following commands:

```
cd html_projects
cvs import -m "Put html_projects in the repository" html_projects vendor release
```

The -m option supplies a log message for the transaction; if you don't use it CVS starts your default editor so you can type a message, then finishes the operation when you exit the editor. The vendor and release tags aren't used by CVS but are required nonetheless. A typical vendor tag could be your company name, and “start” makes a good release tag. If your project contains binary files,

such as pictures, read up on the -k option to ensure they are copied into the repository correctly.

If you were to now look in your repository, you should find a directory called html_projects containing copies of all the files in the original html_projects directory.

Creating a new project under CVS is simply a matter of creating an empty directory structure and then importing it into the repository via the import command. As you create the files for the new project, use the CVS **add** command to put them in the repository.

Using CVS

The basic steps for using CVS are as follows: check a project out of the repository, make changes to project files and verify whether they work, commit the modified files back into the repository and supply notes on the changes you made.

You have to check out files from CVS before you can edit them. By default CVS checks out the latest revision of a project, but you can specify earlier revisions if you wish. When you check out a project from the repository, CVS copies the project's files to the current directory, creating subdirectories as necessary. You can check out project files by specifying a directory name (e.g., html_projects) or by specifying a specific project file (e.g., html_projects/client1/index.html). Specifying a file still creates the project's directory structure in your working directory, but only the specified file is copied from the repository.

To check out client1's files, move to a directory where you can work, such as your home directory, and then issue the CVS **checkout** command shown below. You can easily end up with several copies of a project scattered about if you don't switch to the same starting directory each time you run the checkout command.

```
cvs checkout html_projects/client1
```

Next, switch to the client1 directory (**cd ~/html_projects/client1**) and make changes to the files using your favorite editing tool. Make sure your changes work before committing a file back into the repository. A common mistake when using version control software is to check in files too soon. This causes the repository to contain many versions of the file, most of which don't work.

To put the changed file into the repository, use the following command:

```
cvs commit -m "made some changes" index.html
```

CVS will let you know if the file was successfully placed in the repository and what its new revision number is.

You can retrieve earlier revisions of a file by specifying a revision number or a date with the checkout command. For example, if index.html is currently at revision 1.3 and you want to retrieve yesterday's version, which was 1.2, you can do so with either of the following commands:

```
cvs checkout -r 1.2 html_projects/client1/index.html
```

or

```
cvs checkout -D yesterday html_projects/client1/index.html
```

The -r switch allows you to specify a revision number, while the -D switch allows you to specify a date. You can specify an ISO standard date such as 2000-03-23 or a relative date such as “yesterday”.

Adding and Removing Project Files

To add a file to an existing project, check out the project then create the new file in the project's working directory. Add it to the repository using the following commands:

```
cvs add newfile
cvs commit -m "Added newfile to the project" newfile
```

Removing files is very similar to adding them. First check out the project, and then delete the files you wish to remove from the working directory. Remove them from the repository with:

```
cvs remove newfile
cvs commit -m "Removed newfile from project" newfile
```

Project Aliases

One way to deal with a repository full of project directories is to take advantage of the ability to use aliases in place of directory names in CVS commands. Aliases allow you to use short, meaningful names for projects instead of long directory names. An alias can also be used to group separate projects under a single name so they can all be checked out with one command. Finally, an alias can list specific project files, such as documentation or header files, allowing you to check out small pieces of a project. To use aliases you must edit the modules file in the CVSROOT directory of the repository. This is explained in the CVS documentation.

Tagging Project Files

CVS allows you to supply a symbolic or logical name (e.g., release-1 or beta) to all the files in a project with the **tag** command. Since each file in a project might have a different revision number, a tag provides a way to take a snapshot of the project at a given moment. You can then use the tag with the -r switch when checking out a project to retrieve that snapshot, without having to remember the version numbers of each file in the project. One thing to note is that tags can't contain spaces or periods.

Project Branches

CVS allows you to create branches of a project where each branch contains project code in different states, such as a bug-fix branch and a new features branch. You can work on different branches without affecting the other branches and then you can merge the changes from one branch into another automatically.

For the sake of example, let's assume you're working on a project called FaxMan and have released version 1.0. You tagged the source files in the repository as rel-1-0 upon release, then started working on version 2.0. Then you get complaints of bugs in version 1.0 that have to be fixed. To create a branch of the FaxMan project containing version 1.0 you can use:

```
cvs rtag -b -r rel-1-0 rel-1-0-bugfix FaxMan
```

The **rtag** command assigns a new tag (rel-1-0-bugfix) to the code in the repository. The -b flag means the tag is a new branch, and -r rel-1-0 means this branch contains the code previously tagged as rel-1-0.

To check out and work on the version 1.0 code you would use the following command:

```
cvs checkout -r rel-1-0-bugfix FaxMan
```

To merge the bug fixes with the current FaxMan code you first check out the latest code and then tell CVS to merge the rel-1-0-bugfix code with it. This is done using:

```
cvs checkout FaxMan
cvs update -j rel-1-0-bugfix
```

CVS Clients

There are several CVS clients available so you don't have to manipulate CVS from the command line. **TkCVS** is currently at version 6.4 and requires Tcl/Tk 8.1 or better. **Pharmacy**, which is currently at version 0.2.1, is part of the

GNOME project, while **Cervisia**, at version 1.0 stable, uses the Qt and KDE libraries. Another client that uses the Qt toolkit is **LinCVS**, version 0.3. See the Resources section for pointers to these projects.

Conclusion

While CVS is a very powerful tool that allows far-flung developers to collaborate on projects over the Internet, it is also easy to configure and use on a local machine. If you frequently update files and need to keep track of the changes, CVS allows you to build a record of what changes were made.

Resources

Bubble Reviews of CVS Books

Version Control's Next Version

Ralph Krause is a writer, programmer and webmaster who lives in Michigan. He can be reached at rkrause@netperson.net.

Archive Index Issue Table of Contents

Advanced search

Create User Interfaces with Glade

Mitch Chapman

Issue #87, July 2001

Mitch shows how to use gnome-python's libglade binding to build Python-based GUI applications with little manual coding.

Glade is a GUI builder for the Gtk+ toolkit. Glade makes it easy to create user interfaces interactively, and it can generate source code for those interfaces as well as stubs for user interface callbacks.

The libglade library allows programs to instantiate widget hierarchies defined in Glade project files easily. It includes a way to bind callbacks named in the project file to program-supplied callback routines.

James Henstridge maintains both libglade and the gnome-python package, which is a Python binding to the Gtk+ toolkit, the GNOME user interface libraries and libglade itself. Using libglade binding to build Python-based GUI applications can provide significant savings in development and maintenance costs.

All code examples in this article have been developed using Glade 0.5.11, gnome-python 1.0.53 and Python 2.1b1 running on Mandrake Linux 7.2.

Running Glade

When launched, Glade displays three top-level windows (see Figure 1). The application main window shows the contents of the current Glade project file as a list of top-level windows and dialogs defined in the project file. The Palette window shows the Gtk+ and GNOME widgets supported by Glade. When a widget is selected for editing, the Properties window displays the current values of that widget's properties.

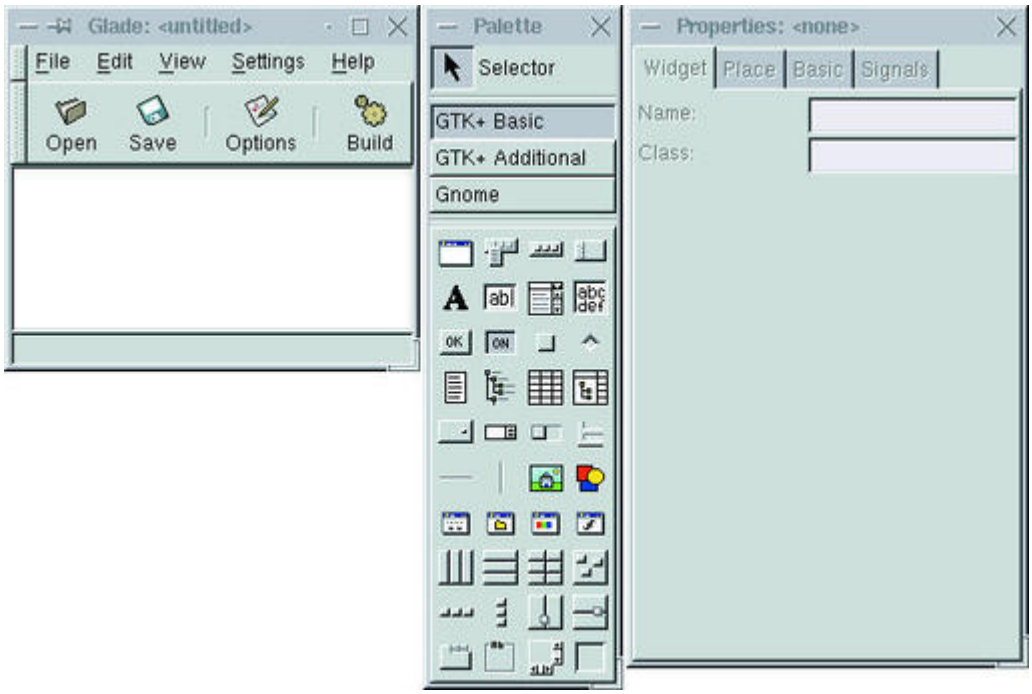


Figure 1. Launching Glade

The Palette window partitions Glade's supported widgets into three groups. “GTK+ Basic” widgets are the most commonly used Gtk+ widgets. “GTK+ Additional” are less frequently used widgets such as rulers and calendars. “Gnome” widgets are taken from the GNOME UI library.

The Properties window displays widget properties in a four-page notebook. The Widget page displays the widget's name along with any properties that are specific to the widget's class. When the widget is placed inside a constraint-based container such as a GtkTable or GtkVBox, the Place page shows the properties that control the widget's placement within its container; otherwise the Place page is empty. The Basic page displays basic properties, such as width and height, possessed by all kinds of widgets. Finally, the Signals page lets you browse the set of Gtk+ signals that the selected widget can emit and lets you bind signal handler functions to those signals.

Creating Widget Hierarchies

The process of laying out a widget hierarchy within Glade is similar to that in environments such as Visual Basic. The root of every hierarchy is a top-level window or a dialog. Widgets can be placed within these top-level containers by first selecting, in the Glade Palette window, the type of widget to be created, then clicking on any cross-hatched region within the containers.

Defining Signal Handlers

The Signals page of the Glade Properties window lets you add application-specific behavior to your application. The top part of the page lists the signal

handlers defined for the current widget. The controls in the bottom part of the page let you select one of the signals emitted by the widget and create a new handler for it.

To define a new signal handler, click on the ellipsis button to the right of the Signal entry field. A Select Signal dialog appears, listing all of the signals that this widget can emit. The signals are grouped by the Gtk+ widget class in which they are defined (see Figure 2).

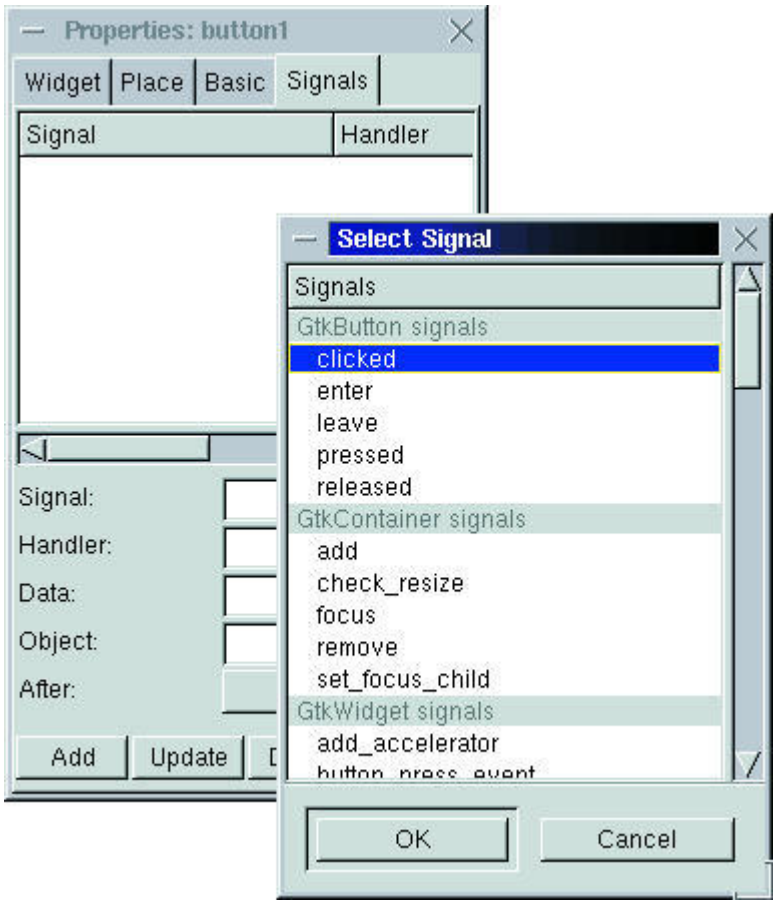


Figure 2. Select Signal Dialog

Once you have selected a signal, click OK in the Select Signal dialog. The name of the selected signal appears in the Signal entry field of the Signals page. Glade also automatically fills in the Handler field, using the naming convention of "on_<*widget*>_<*signal*>". You can change the name manually if Glade's naming conventions don't suit your needs.

The bottom portion of the Signals page provides additional entry fields where you can supply application-specific data, specify an object to receive the signal, and so on. I always leave these fields empty because they aren't needed when working with gnome-python.

Glade Project Files

Glade saves information about a project in an XML-formatted project file having a filename extension of .glade. Glade's use of XML makes it easy to build separate add-on tools that operate on project files, such as code generators for new programming languages.

The first time you save a new project, Glade presents you with a Project Options dialog. Most of the settings in the Project Options dialog matter only when you are using Glade to generate source code for your project. However, some settings, such as the project directory, are important even when you are just using Glade as a layout tool.

By default Glade assumes you want to save your new project under your login directory, in a subdirectory named Projects/project1. This is probably not what you want. I usually save the project in the directory in which Glade was started.

Fortunately, it's easy to reset the project directory. Just click the Browse... button to the right of the Project Directory entry field, and a dialog entitled Select the Project Directory appears. This dialog selects Glade's current working directory by default, so you can just click its OK button.

When you do so the Project Directory field in the Project Options dialog changes to the current working directory, and the Project Name field goes blank. Type in a new project name, and the Program Name and Project File fields update accordingly (see Figure 3). When you click OK, your project will be saved to the specified project file.

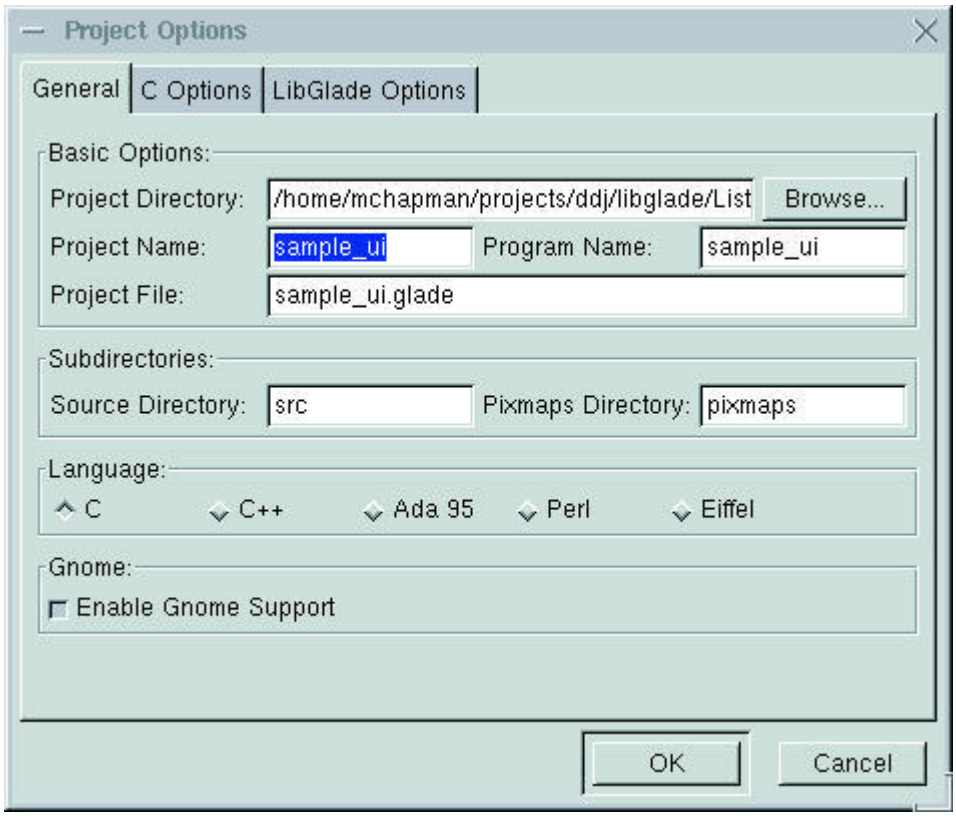


Figure 3. Project Options Dialog

Using libglade

Once you have created a Glade project file, you can use gnome-python's libglade module to create the visual hierarchy described in the project file and to gain programmatic access to the widgets in the hierarchy:

```
import libglade
loader = libglade.GladeXML ("helloworld.glade", "window1")
```

The libglade library defines a class, GladeXML, which does most of the work. To load a widget hierarchy, instantiate GladeXML and pass it the name of the Glade project file, along with the name of the topmost widget that you want to load from the file.

Note that you can supply the name of any widget in the hierarchy, even if it's buried deeply within a top-level window. This makes it possible to partition complex visual hierarchies—for example, the pages of a complex notebook-based interface—across multiple Glade project files. It also makes it easy to handle projects with dynamic visual content, loading only the components that are appropriate at any given time.

Once you have loaded a widget hierarchy, GladeXML lets you look up specific widgets by name via the **get_widget** method. **get_widget** returns the widget you requested or “None” if the widget cannot be found:

```
window1 = loader.get_widget("window1")
if window1:
    window1.set_title("Hello, World!")
```

Connecting Signal Handlers

One of the most powerful features of GladeXML is that it can bind Python callable objects (methods, functions, etc.) to signal handlers named in a Glade project file. The **signal_autoconnect** method makes this possible.

signal_autoconnect takes one argument: a dictionary mapping signal handler names to Python callables. For each of the signal handlers you've defined in your Glade project file, signal_autoconnect looks up the corresponding Python callable in the supplied dictionary. If a matching entry is found it is bound to the signal. In other words, your Python callable gets installed as the signal handler:

```
def button1_click_handler(*args):
    print "Don't push that button!"
signal_handlers = {
    # Exit the main event loop when the user closes
      the main window.
    'on_window1_delete_event': gtk.mainquit,
    # Call button1_click_handler when the user clicks
      button1.
    'on_button1_clicked': button1_click_handler
    }
loader.signal_autoconnect(signal_handlers)
```

GladeBase

By itself, libglade greatly reduces the manual coding needed for a gnome-python application. Widget hierarchies can be laid out using Glade and loaded with just two or three lines of code, as opposed to the hundreds that would be needed to create them using direct pygtk calls. What's more, behaviors can be added simply by assembling a dictionary of Python callables and passing it to GladeXML.signal_autoconnect instead of repeatedly invoking widget connect methods.

libglade saves a lot of effort, but it could do more. For instance, large Python applications are often structured as a small main program and an associated collection of Python packages installed somewhere on the Python path. Maintenance costs would be reduced if the application's Glade project files could be stored together with its Python packages and “imported” at runtime via relative pathnames.

It would also be helpful if widgets could be accessed directly as instance variables of some sort of UI hierarchy object without having to be located via GladeXML.get_widget.

Finally, it should be possible to automate building a dictionary of the callables in an object's namespace and passing that dictionary to signal_autoconnect.

This would allow clients to define signal handlers as object methods and avoid explicitly registering the handlers.

The following sections describe a module, GladeBase, that provides these features. GladeBase also recasts the services of libglade to fit the MVC (model view controller) design pattern (see Listing 1 at ftp://ftp.linuxjournal.com/pub/lj/listings/issue87/). GladeBase has two principal exports: class UI and class Controller.

GladeBase.UI

GladeBase.UI corresponds to the View component of the MVC design pattern. It is responsible for creating a widget hierarchy from a Glade project file and for updating the visual content of an application under direction of an associated controller. GladeBase.UI is derived from libglade's GladeXML class, so it inherits all of the methods discussed earlier.

The GladeBase.UI constructor takes three arguments: the filename of the Glade project file from which it will load its widget hierarchy, the name of the widget that serves as the root of the hierarchy and an optional keyword argument, gladeDir, which is the relative pathname of a directory in which to look for Glade project files.

The gladeDir keyword argument defaults to the current working directory. It is joined with the filename argument to form the relative pathname of the Glade project file.

It may seem odd to use both gladeDir and filename parameters instead of specifying the location of the Glade project file with a single relative pathname. But this separation can reduce maintenance costs for any application that stores its Glade project files in a single subpackage.

Such an application can define a subclass of GladeBase.UI, which provides a hardwired value for gladeDir:

```
import GladeBase
class UIBase(GladeBase.UI):
  def __init__(self, filename, rootname):
    GladeBase.UI.__init__(self, filename, rootname,
                          gladeDir="MyApp/GladeFiles")
class MainWinUI(UIBase):
  def __init__(self):
    UIBase.__init__(self, "main_win.glade", "window1")
```

Then the application can derive all of its UI classes from this subclass. In this way the application can specify in one place the relative pathname of the directory containing all of its Glade project files.

A helper module, PathFinder.py, enables GladeBase.UI to search the Python path for files. The PathFinder.find function takes a pathname as its sole argument. If the pathname is absolute, it is returned without further processing. If it is a relative pathname, the find function joins it with each Python path entry in turn to create a candidate pathname. If the candidate pathname exists, it is returned. If no candidate pathname matches, find raises a PathFinder.Error exception (see Listing 2).

Listing 2. PathFinder.py

The GladeBase.UI.__getattr__ method makes it possible for clients to access the widgets in a GladeBase.UI hierarchy as though they were attributes of the instance. The __getattr__ method assumes that the attribute name provided by the caller is the name of a widget and looks up the widget using GladeXML.get_widget. Once the widget is found, it is cached as a new instance variable to speed up future access. If the requested widget can't be found, __getattr__ raises an AttributeError.

If a widget hierarchy contains more than one widget with the same name, there's no telling which one will be returned by GladeBase.UI. When you're using GladeBase.UI it's a good idea to name widgets the same way you would name Python instance attributes: each name should be unique to the object and should be a valid Python identifier.

Application-specific UI classes usually extend GladeBase.UI with methods to perform complex user interface updates.

GladeBase.Controller

GladeBase.Controller corresponds to the Controller component of MVC. A Controller responds to user input events by translating them into changes in the state of the application data model. Similarly, it responds to changes in the data model by translating them into UI updates.

GladeBase.Controller doesn't help you respond to changes in your application's data model, but it does automatically wire up signal handler methods to the signal handlers defined in a Glade project file.

The GladeBase.Controller constructor takes one argument: an instance of GladeBase.UI that is the UI to be controlled. During initialization, a new GladeBase.Controller instance traverses its class hierarchy, building up a dictionary of all callable objects in the instance's namespace (the traversal starts with the instance dictionary in case any callables have been defined as instance attributes). GladeBase.Controller then passes this dictionary to the signal_autoconnect method of the supplied GladeBase.UI instance.

Application-specific controller classes extend GladeBase.Controller simply by defining signal handler methods:

```
class Controller(GladeBase.Controller):
    def __init__(self, ui):
        ...
        GladeBase.Controller.__init__(self, ui)
    def on_window1_delete_event(self, *args):
        gtk.mainquit()
    def on_button1_clicked(self, *args):
        print "Button 1 clicked."
```

Generating Controller Stubs

GladeBase automates the conversion of Gtk+ widget hierarchies to Python object hierarchies and automatically connects Python-based signal handlers, but it still requires you to identify and implement all of the signal handlers defined in a Glade project file. For pure Gtk+ projects this is no problem because the only signal handlers are the ones you explicitly define.

However, when you use Glade to build a GNOME application, many signal handlers are defined automatically. For example, a new Gnome Application window is created with a standard menubar whose menu items all have predefined activate signal handlers. It can be tedious to browse through GNOME-based projects, manually locating predefined signal handlers and adding them to your application controllers.

As noted earlier, Glade project files are stored in an XML format (as of yet there is no DTD describing the structure of a project file, but it is easy to understand by inspection). Python 2.0 includes an XML library, layered atop James Clark's Expat library. So it's fairly easy to build a Python application that rummages through a Glade project file, identifies all of the signal handlers declared in a given widget hierarchy and generates a stubbed Controller module for that hierarchy.

GladeProjectSignals.py (see Listing 3 at ftp://ftp.linuxjournal.com/pub/lj/listings/issue87/) extracts signal-handler information from a Glade project file. The module has two main abstractions. Class WidgetTreeSignals traverses an XML DOM (document object model) tree representing a widget hierarchy and records all of the signal handler declarations it finds. Class GladeProjectSignals loads a Glade project file and builds up a dictionary of WidgetTreeSignal instances, one for each top-level widget defined in the project file.

The constructor for WidgetTreeSignals takes a DOM node as argument. It assumes this node describes a widget and expects it to contain a name node defining the widget's name. Having recorded the widget's name, WidgetTreeSignals walks the DOM tree. It checks each visited node to see if it is a signal node. If it is, WidgetTreeSignals records the value of the node's handler

child, which should be the name of a signal handler. Otherwise, WidgetTreeSignals assumes the node contains child nodes and continues traversing those.

GladeProjectSignals is comparatively simple. It uses Python's xml.dom.minidom package to load a Glade project file as a DOM tree. Then it searches the tree for top-level widget nodes (a Glade design file contains other top-level nodes such as the GTK-Interface and project nodes). For every widget node found, GladeProjectSignals creates a new WidgetTreeSignals instance, which in turn lists the signal handlers defined by that widget and its descendants. Each WidgetTreeSignal instance is stored in a dictionary, self.widgets, keyed by top-level widget name.

ControllerGenerator.py (see Listing 4 at ftp://ftp.linuxjournal.com/pub/lj/listings/issue87/), when invoked with a Glade project filename and the name of a top-level widget defined in that file, prints out a stubbed Controller for that widget and its children.

Most of the module's work is done by class ControllerGenerator. This class defines a generate method that takes a Glade project filename and top-level widget name as arguments. The generate method uses an instance of GladeProjectSignals to find the handlers for the named widget. Then it creates a list of stubs for those handlers. Using a template string and Python's string formatting operators, generate produces a string containing the body of the stubbed Controller module and returns that to its caller.

Conclusion

Glade, libglade and gnome-python can greatly reduce the effort of building Gtk+ and GNOME applications in Python. The tools presented in this article reduce maintenance costs even further by automating the conversion of Glade widget hierarchies to Python object hierarchies, automatically connecting signal handlers defined in Controllers and generating stubbed Controllers.

Resources

Mitch Chapman (chapman@bioreason.com) is a senior software engineer at Bioreason, Inc. He lives in Santa Fe, New Mexico where he can enjoy Python

programming, snowboarding, rock climbing, squinting into the sun and flying more or less simultaneously.

Archive Index Issue Table of Contents

Advanced search

Automating Firewall Log Scanning

Leo Liberti

Issue #87, July 2001

Techniques and scripts for automating scanning of log files produced by ipchains.

Firewalls are computers dedicated to filtering particular kinds of network traffic between two networks. They are usually employed to protect a LAN from the rest of the Internet. Securing every box on the LAN is much more costly and time consuming than deploying, administering and monitoring a single firewall. A firewall is particularly essential to those institutions permanently connected to the Internet. Depending on the network configuration, the router can be set up as a packet filter; usually, though, it is more convenient to set up a dedicated box to act as a firewall. Because they can be made extremely secure and have a low cost, Linux boxes can be very effective firewalls.

Deploying a firewall on the Linux kernels 2.2.x is done with **ipchains**, while **iptables** are used on the new 2.4.x kernels. How to set up the actual firewall is beyond the scope of this article; we refer the reader to the ipchains HOWTO for the 2.2.x kernels and to Paul “Rusty” Russell's Packet-Filtering HOWTO for the 2.4.x kernels. Both of them can be found on the Internet by using any search engine. But building the actual firewall is not enough; in order to offer tight security, a firewall needs to be monitored. In this article we explain how to build and use a web-based ipchains monitoring system called **inside-control**.

There are two main uses of a firewall monitoring system: to check that no malicious cracker is trying to wreak havoc in the internal LAN and to check that users inside the LAN are not abusing the internet service.

Firewall Setup Example

Here is a setup for a very simple firewall to which we will refer as a working example later in the article.

Suppose, for example, that the internal network is 10.0.1.0/255.255.255.0, the Linux gateway/firewall has the addresses 10.0.1.1 on the interface connected to the internal LAN and 10.200.200.1 on the interface connected to the Internet (both IP addresses are in fact nonroutable, so this is just a fictitious example). The first step to setting up a firewall is to enable gatewaying between the network interfaces:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

We then proceed to build up a logging firewall using ipchains. First we flush all preceding rules, and we allow packets on the loopback interface and all ICMP packets:

```
ipchains -F
ipchains -A input -i lo -j ACCEPT
ipchains -A input -p ICMP -j ACCEPT
```

Now we block and log the Telnet protocol from the Internet to the internal LAN:

```
ipchains -A input -p TCP -s 0.0.0.0/0 -d 10.0.1.0/24 23 -l -j DENY
```

But we allow and log the HTTP protocol from the internal LAN to the Internet:

```
ipchains -A input -p TCP -s 10.0.1.0/24 -d 0.0.0.0/0 80 -l -j ACCEPT
```

Finally we set up permissive policies:

```
ipchains -P input ACCEPT
```

This firewall blocks and logs all incoming Telnet connections, it allows and logs all outgoing HTTP connections, and it allows everything else (see Figure 1). Such a setup is too permissive for serious protection, but it will illustrate well what the automated log scanning script can do.

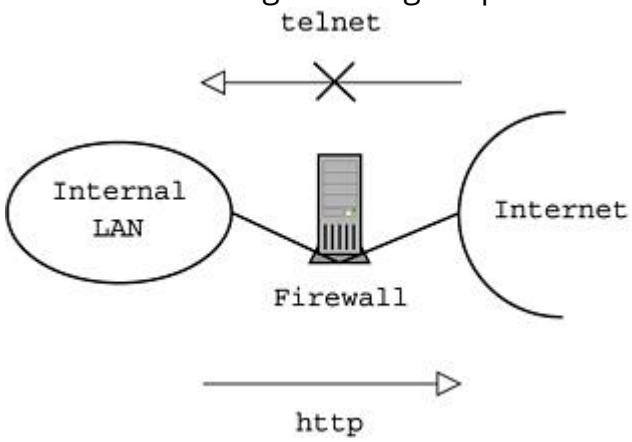


Figure 1. Setup of Sample Firewall

The file the firewall outputs its logs to is usually either /var/log/syslog or /var/log/messages. In order to find out which one, you can do

```
grep -q "Packet log" /var/log/syslog && echo yes
```

If it outputs “yes” then it is /var/log/syslog, if it outputs nothing it is most probably /var/log/messages. You can confirm with

```
grep -q "Packet log" /var/log/messages && echo yes
```

If both commands produce no output, then the firewall is inactive or there was no logged traffic (in our example, Telnet and HTTP) through the firewall.

2.4.x Kernels and iptables

Regarding the 2.4.x kernels and iptables, things are a bit more complicated. First you must remember to compile the kernel with all of the packet-filtering options, including the LOG target. Second, change ipchains to iptables. Then change the names of the chains to uppercase (e.g., input becomes INPUT). Next, change the name of the targets (DENY becomes DROP). Lastly, specify port numbers in a different way. Listing 1 is the 2.4.x sequence of commands equivalent to the 2.2.x sequence of commands given above.

Listing 1. 2.4.x iptable Command Sequence

ipchains Log Format

Let us now examine a sample log entry from our firewall's /var/log/syslog:

```
Jun 12 16:15:54 myfirewall kernel: Packet log: input DENY eth1 PROTO=6 212.65.214.2:34251 10.0.1.2:23
```

This means that at quarter past four in the afternoon on 12 June, the firewall (called, rather boringly, myfirewall) denied and logged a packet coming into its network interface eth1 (the external interface on the Internet) with the TCP protocol coming from 212.65.214.2 (from port 34251), directed to 10.0.1.2 (on port 23, i.e., the Telnet port) and having a length of 52 bytes. We shall skip most of the other details, apart from one: “SYN” means that the packet is the first packet of a connection. In practice, this information is very useful in discriminating those packets that are part of a pre-existing connection (that might have been initiated from the internal LAN) and those packets that attempt to establish a connection from the Internet towards the internal LAN. Usually one allows “reply” packets (which do not have the “SYN” flag set) but denies “SYN” packets because it means somebody out there is trying to make a connection to a computer in the internal LAN.

Of course, it is possible to check the status of a firewall by inspecting all relevant entries in the log file, but this is feasible if one logs only a few strange-looking packets. For example, on some firewalls I set up I decided to log all those packets coming from the Internet towards port 31337 on computers on the internal LAN, as 31337 is the default port BackOrifice uses. Whenever one is interested in getting some statistics from the firewall, it is likely that the size of

the log file will be in excess of 5MB per day. In such cases, inspection of the log file by hand is no longer an option. This is when automated log scanning comes in.

When analyzing 2.4.x kernel firewall logs, the format is different:

```
Jun 12 16:15:54 myfirewall kernel: Packet log: IN=eth1 OUT= MAC=00:00:00:00:00:00:00:00:00:00:00:00:08
```

The fields we are interested in are SRC (source IP address), DST (destination IP address), SPT (source port), DPT (destination port) and the presence or absence of the SYN flag.

The inside-control Script Structure

I am going to use Perl to build the log scanner. It is not the only option and, in fact, in order to achieve top performance one should use a compiled language. When I recoded this script in C++, I observed an execution speed gain of 100%.

The **inside-control** script is composed of a main parsing loop and an HTML data display loop. Since the script is a CGI it needs to reside on a web server configured for running CGI programs.

Note that the code, as described below, sacrifices functionality and useful details like error-checking for clarity. For example, there is no check that "opening a file" was successful before actually reading that file. Note also that the code below is customized to analyze the packet-logging format of kernels 2.2.x. Changing to the logging format of kernels 2.4.x, on the basis of the sample packet log described above, should be straightforward.

Main Parsing Loop

First, we open the log file and initialize some variables (those with Red Hat should use /var/log/messages instead of /var/log/syslog):

```
#!/usr/bin/perl
open(LOGFILE, "/var/log/syslog");
$firstdate = "";
$date = "";
$total_traffic = 0;
```

Now we loop over each line in the log file:

```
while ( <LOGFILE> ) {
```

Skip all log entries which do not belong to the firewall:

```
next unless /Packet log/;
```

We also parse the line (warning: in the Perl script, write the last line in this chunk as a whole long line, without the backslash):

```
chomp;
@log = split;
($month,$day,$time,$policy,$proto,$ipsource,$ipdest, \
$tot_len) = @log[0,1,2,8,10,11,12,13];
```

We then calculate the date and store the first date in the log. As we go on, we store the current date as the last date, so that after the last step the variable lastdate will contain the last date in the log:

```
$date = $day . " " . $month . " " . $time;
if (length($firstdate) == 0) {
  $firstdate = $date;
}
$lastdate = $date;
```

Read the protocol type, the source IP address, the source port, the destination IP address, the destination port and the packet length:

```
$proto = substr($proto, -1);
($ips, $ports) = split ":", $ipsource;
($ipd, $portd) = split ":", $ipdest;
($flush, $packetlen) = split "=", $tot_len;
```

Now record the destination IP address in a string, and associate that string to the source IP address so that in the data display loop we will be able to loop over source IP addresses and retrieve the hosts they connected to:

```
unless ( $sourcedest{$ips} =~ /$ipd/ ) {
  $sourcedest{$ips} = $sourcedest{$ips} . $ipd . " ";
}
```

We count the log entries for the source IP address:

```
++$source{$ips};
```

and sum up the total traffic volume:

```
$total_traffic += $packetlen;
```

Finally, we sum up the per-host traffic volume:

```
$traffichost{$ips} += $packetlen;
}
```

Notice that not all the information gathered has been used (no talk of ports, for example), so there is plenty of room for expansion here.

Data Display Loop

First we display a nice-looking web page header, as shown in Listing 2.

Listing 2. Web Page Header

Loop over the sorted source IP addresses and print the source IP address, the number of packets coming from that IP and the traffic (in bytes) generated from that IP:

```
for (sort keys %source) {
  print "<TR><TD>$_</TD> ";
  print "<TD>$source{$_} </TD>\n";
  print "<TD>$traffichost{$_} bytes</TD>\n";
```

Now we are able to print the string containing the destination IP addresses contacted by the current source IP address:

```
$tmp1 = $sourcedest{$_};
if (length($tmp1) gt 0) {
  print "<TD>\n";
  @lt1 = split " ", $tmp1;
  for(sort @lt1) {
    printf "$_ <br>\n";
  }
  print " </TD>\n";
}
print " </TR>\n";
}
```

Finally, we print the HTML tail:

```
print "</TABLE>\n";
print "</center>\n";
print "</BODY></HTML>\n";
```

The Downloadable inside-control Script

The version of inside-control I actually implemented is richer in functionality than the one presented here. You can download the script from www.iris-tech.net/hdsl-fw. Some of the main added features include the ability to display arbitrary names instead of IP addresses in the “Source IP” column. This is done with a very simple text database that maps IP numbers to names. The format is the same as the /etc/hosts file, and you can use that file if it is meaningfully configured for your internal LAN. The exact location of the “IP to names” database file can be specified by changing the relevant variable (*$useripdb*) at the beginning of the script.

There is also a search facility that allows one to look for a particular source IP address (or corresponding name found in the “IP to names” database) in the logs. The search form is displayed whenever the CGI is called without arguments from the browser. Arguments passing is done by the GET method.

Additionally, the main loop includes some data validation (the kernel cannot always log properly, especially on low RAM or low-spec CPUs) and some storage of port-dependent information.

Finally, the script can also be called without the web interface. Just pass any argument to inside-control, and all HTML output will be suppressed and some

normal output will be provided instead. A search string for a source IP address (or its corresponding name found in the “IP to names” database) can be passed to the program via the -t option.

Notes and Caveats

The purpose of this article is to explain some design principles and give some hints, not to give a prepackaged solution to log scanning problems. There are many areas where the inside-control script can be made better, such as performance and security. The following are some notes about inside-control, mostly related to security issues.

In order for a CGI to read the computer log files /var/log/syslog or /var/log/messages, these have to be made readable by all. This can be accomplished with the command **chmod +r /var/log/syslog**. This, however, is not very secure as it gives anybody on the system permission to read the computer log files. It would be much better to get the web server to run inside-control with a particular group permission, and then make the log files belong to that group.

After reading the article, one could conclude it is essential that a firewall also runs a web server, as inside-control needs to read the firewall log files. In fact, putting a web server on a firewall is very insecure: ideally a firewall should run no dæmon service, and all maintenance should be done at the console. When there is a need for remote administration, the only service that may be installed on the firewall is **ssh**, the secure shell. Running inside-control is still possible by setting up a separate web server within the internal network that also acts as a syslog server for the firewall.

Firewall logs can fill up a partition pretty quickly. In order to avoid having a clogged hard disk on the firewall (which could lead to a malfunctioning internet connection), depending on the amount of traffic you want to log, you have to allow for a large log file space. For high data volume services (typically HTTP, FTP, SMTP, NetBIOS, LPD and database services) I would advise setting up a second hard disk of at least 20GB in size, with just one partition mounted on /var/log. Also keep in mind that the script needs some error-checking code on critical steps like opening a file.

Finally, there is a lot of room for improvement everywhere in the script and especially in the main loop. One can use much more data from each log line than is discussed here. However, it is always a good idea to not show too many details; otherwise, the whole point of having an automated log scanner is defeated. If you display all available details, you end up having to look for suspicious entries in an unmanageably high volume of traffic log.

Leo Liberti is technical director at IrisTech in Como, Italy, a firm that supplies its customers with web-based applications and all kinds of electronic services. His free time is dedicated to eating in as many restaurants as possible.

LINUX JOURNAL

Custom JSP Actions

Reuven M. Lerner

Issue #87, July 2001

Learning shorthand for complicated Java code.

Over the last few months, we have looked at server-side Java from a number of perspectives. We began with servlets, Java classes that are executed from within a servlet container. While programmers are not especially daunted by servlets, graphic designers might feel otherwise.

Solving this problem are JavaServer Pages (JSPs) that combine Java and HTML, using a syntax similar to Microsoft's Active Server Pages (ASP) or the open-source HTML::Mason system for mod_perl. Each JSP is really a servlet in disguise; the JSP engine translates the page into a servlet, and then compiles the servlet into a Java .class file.

JSPs can include straight Java code that can make it easier to perform complex actions. But at a certain point, this code can overwhelm the HTML, making it impossible to maintain the JSP. Nonprogrammers are also turned off by large amounts of code inside a JSP, defeating much of the purpose of using JSPs instead of straight servlets.

Last month, we looked at one way to avoid putting code inside of JSPs using JavaBeans. Using simple XML-based tags, a nonprogrammer can put together JSPs that exhibit complex behaviors, without having to write a single line of code. Indeed, the real magic of JavaBeans is not the beans themselves, but rather the special tags that allow us to work with them so easily.

This month, we will learn how to write our own “custom actions”, as they are known—XML-based tags that allow us to work with Java classes and methods without having to work with Java itself. Our examples are designed to work with the open-source Jakarta-Tomcat implementation of servlets and JSPs. However, they should work with any JSP implementation that works with custom actions.

There are a number of reasons to use custom tags. First of all, they reduce the amount of Java code we must put inside of our JSPs, making them easier to read, understand and maintain. In addition, custom tags are less complicated than Java code, making them suitable for a wider audience than Java code users. Finally, each custom tag library points to a centrally written and maintained Java class. Using custom actions, a site can thus create a library of tags appropriate for its particular needs. One Java programmer can create and publish a tag library for a number of graphic designers and JSP authors. As we will see, custom tags aren't a panacea, but they can be quite useful, and I consider them one of the most compelling reasons to use JSP over competing technologies.

What Are Custom Actions?

Custom actions provide us with a shorthand for complex Java code within our JSP. Anything you do with a custom action could also be accomplished with Java code inside of the scriptlet (<% %>) tag. After all, the JSP is turned into a servlet before it is compiled and executed for end users.

As we saw last month with JavaBean tags, custom actions are defined with XML rather than HTML. This can be confusing and frustrating at first, especially for those of us who have acquired bad habits when writing HTML. The following might appear to be legal:

```
<P><jsp:getProperty name="simple"
property="userID"></P>
```

But in fact, the above line will not work and will result in an exception and stack trace within the JSP. That's because all tags in XML must be closed somehow. If a <tag> has no matching </tag>, then it must indicate that it closes itself with <tag/>. Thus, the above line must actually be written as:

```
<P><jsp:getProperty name="simple"
property="userID"/></P>
```

Custom actions are merely syntactic sugar for Java methods. Each tag library defines a set of actions. For example, the jsp tag library defines three actions: “getProperty”, “setProperty” and “useBean”. Each action is defined by a single Java class, known as a tag handler.

To define a tag library, we create an XML file known as a tag library descriptor, or TLD. The TLD connects each action to its appropriate tag handler class, listing optional and mandatory attributes, as well as other information about the tags.

To use our custom actions within a JSP, we use a special directive to load our TLD. This helps the JSP engine to validate the custom tags within our JSP, as well as to find the tag handler class associated with these actions.

A Simple Custom Action

We will now define one simple custom action in order to understand the underlying mechanics of working with tag handler classes, TLDs and JSPs.

Our custom action will be a “hello” tag, which takes an optional “firstname” parameter. If the parameter is there, our tag will produce a simple “hello” message to the named user. If the parameter is missing, our tag will produce a generic “hello” message.

The first step is to write a simple tag handler that will implement this functionality. Such a tag handler is shown in Listing 1, defining the HelloTag class. I put the HelloTag.java source file, along with all JSP- and servlet-related classes, under the $TOMCAT_HOME/classes directory. Since HelloTag.java is in the il.co.lerner package, and since $TOMCAT_HOME on my machine is /usr/java/jakarta-tomcat-3.2.1, this means I placed my Java source file in:

/usr/java/jakarta-tomcat-3.2.1/classes/il/co/lerner/HelloTag.java

Listing 1. HelloTag Tag Handler

After compiling HelloTag.java into HelloTag.class, this tag handler can be incorporated into one or more tag libraries.

Each tag handler class must implement one of two different standard interfaces, Tag or BodyTag. (The latter is for custom actions that have a body between their opening and closing tags, rather than those we will discuss this month, which have no body.)

In practice, there is no reason to implement these interfaces. It is easier and more practical to inherit from the TagSupport and BodyTagSupport classes, which provide default implementations for the interfaces. By subclassing TagSupport, we can save ourselves some work, overriding only those methods for which we don't want the default behavior. In the end, our implementation of HelloTag requires only three methods: setFirstname, doEndTag and release.

The first method, setFirstname, looks and acts just like a JavaBean property-setting method, taking a single argument and returning void. setFirstname is invoked automatically when the JSP engine encounters our custom action with a “firstname” parameter. The parameter value is set to the value passed in the

tag. As with JavaBeans, the method that sets firstname must be named setFirstname, with a capital “F”.

Our second method, doEndTag, is invoked when the JSP engine encounters our custom action's closing tag. The doEndTag method takes no arguments and returns an integer. But instead of returning an integer, we will return one of the symbolic constants provided for us. Normally, we will return EVAL_PAGE, which tells the JSP engine that it should continue to evaluate the remainder of the JSP from which our custom action was invoked. If we wish to stop the JSP engine from evaluating the file any more, either because we have encountered an error or because we want to forward the user to another URL, we can return SKIP_PAGE instead.

Inside of doEndTag, we can place any Java code we might like. In addition to any instance variables we create, we have access to information about the JSP itself, including its HTTP request and response. This is how we can write information to the user's browser, replacing the custom tag with HTML, XML or plain text. (Custom actions generally return plain text, allowing the JSP author to choose how that text will be formatted.) Using the PageContext object, defined by our TagSupport superclass, we can retrieve an output stream and send data to it:

```
   pageContext.getOut().println("Hi there!");
```

Finally, we define the release method, which takes no parameters and returns void. release() is invoked when the custom action has finished execution, and it gives the tag handler class a chance to clean up after itself. In general, this means setting each of the instance variables to null, but it might also involve closing a connection to a relational database or sending information to the error log. In HelloTag.java, we simply assign firstname the null value, and then ask our superclass to nullify each of its own values.

Now that we understand each of the individual methods in HelloTag, how do they work together? When a JSP contains a custom action mapped to our class (via the TLD, described below), each of the action's parameters invokes a “set” method in our class. For example, someone passing the parameter firstname=“foo” will effectively invoke setFirstname(“foo”).

Since we want to make firstname an optional parameter, we give it a default value (null) when we first create it. When the JSP engine finishes evaluating our custom action, it invokes doEndTag and looks at the value of firstname. If firstname is null, it sends a generic (“Hi there!”) message to the end user. If firstname is non-null, however, doEndTag uses its value to send a more personal message to the end user.

When the custom action has finished executing, the JSP engine invokes release(), resetting firstname and a number of other objects.

Writing the TLD

Once we have written our class, we can write a TLD that describes it to the JSP engine. Many people might prefer to work in the opposite direction, using the TLD as a specification JSP authors and tag handlers can use while working in parallel. I prefer to write the custom actions first, modifying the TLD as I go along, even though this is admittedly not the safest nor the most elegant means of working.

The TLD, as you can see from Listing 2, can be a relatively short XML file. The TLD maps action names to the classes that implement those actions. A TLD can map a single action to a single class, or it might map hundreds of actions to hundreds of different classes. And because each class exists separately, it is even possible (though hardly a good idea) for a class to be used in multiple TLDs simultaneously.

Listing 2. hello.tld

The TLD is loaded into our servlet container when it is first referenced. Unfortunately, this means that changing the TLD after the custom action has already been invoked requires restarting Tomcat (and Apache, if you are using Apache's mod_jk along with the Tomcat server). It tells the JSP engine which versions and specifications your tag library supports, making it possible for a JSP engine to know when a particular library needs to be upgraded in order to be compatible with current standards.

The TLD consists of a top-level <taglib> tag, which contains a minimum of four sections: <tlibversion> indicates the version of the tag library specification this library supports; <jspversion> indicates the version of the JSP specification for which the tag library was written; <shortname> gives this tag library a name, which some JSP engines use; and <tag> appears once for every tag handler class we want to include in our library. Each tag gets its own name, the name of the action that is invoked. Thus, if we import a tag library with a prefix of “abc”, the tag named “hello” will be invoked as “abc:hello”. The <tagclass> section maps the tag's name to the tag handler class that actually performs the actions; this class must obviously be in your server's CLASSPATH. The <info> section allows us to provide some basic information and in-line documentation about this particular tag.

Finally, we name each of the attributes this custom action takes. Each attribute has its own <name> tag, as well as an indication of whether the attribute is required.

Using Custom Actions in a JSP

Now that we have a TLD and a tag handler class, we can use them together in any of our JSPs. We import the tag library using the special JSP taglib directive:

```
<%@ taglib uri="/WEB-INF/hello.tld"
prefix="hello" %>
```

Notice how the taglib directive takes two parameters, “uri” and “prefix”. The uri portion contains the filename of the TLD that we just created. If you want to put TLDs directly inside your WEB-INF directory, then the above syntax is perfectly valid. The prefix parameter is a sort of namespace declaration, telling the JSP engine what prefix we will attach to each of the actions the tag library imports. Giving the JSP the option of naming the prefix, rather than building it into the tag library itself, allows us to import multiple tag libraries without having to worry about namespace clashes.

Since our TLD defines a single “hello” tag, and since we imported the tag library using the “hello” prefix, we can invoke our HelloTag methods using the following syntax: **<hello:hello/>**. Listing 3 contains a complete JSP (test-tag.jsp) that demonstrates how we can use this tag.

Listing 3. test-tag.jsp

Remember to include the trailing slash when invoking custom actions. If you forget to include it, Tomcat's JSP engine (known as Jasper) will produce an error message similar to the following:

```
Unterminated user-defined tag:
ending tag &lt;/hello:hello&gt; not found or
incorrectly nested
```

Our TLD indicates the firstname attribute is optional. If we don't pass a firstname parameter, then we get the following output in our web browser:

```
This is a test of our custom action.
Hi there!
```

We can also pass an optional firstname parameter:

```
<hello:hello firstname="Reuven"/>
```

If we put the above in our JSP, the following output is sent to the browser:

```
This is a test of our custom action.
Hello, Reuven
```

More Advanced Custom Actions

The above is a trivial example of how custom actions work. Custom action tags can do much more than simply print names. For example, objects can connect to a relational database, retrieving (or storing) information without requiring explicit Java inside of our JSPs. Custom actions can also act as iterators or provide us with conditional execution.

In order to perform these more advanced actions, we will take advantage of the fact that a tag handler class can look at the body of a custom action; that is, whatever text might happen to reside between the action's opening and closing tags. We can do all sorts of things with this text, ranging from iteration and conditional execution to asking the JSP engine to evaluate its contents before passing it to the tag handler. It is even possible to nest one tag inside of another, effectively passing values from one action to another.

There are a number of open-source tag libraries, including one provided by the Jakarta project itself, which use these functions to provide a great deal of functionality in a number of tags.

Are Custom Actions a Good Thing?

Custom actions are an extremely powerful tool. They provide a wealth of advantages over putting straight Java code inside of JSPs, encapsulate complex behavior inside of easy-to-remember tags, make it relatively easy for nonprogrammers to work with databases and other nontrivial systems.

But there is a problem with custom actions that can be traced back to the word “custom”. The ability to define your own tags within JSPs is a clever and sophisticated tool and provides a number of benefits to everyone involved in developing a web site. However, part of the beauty of the Web is that it is relatively standardized.

Moreover, custom actions can be used to create an entirely new language written in Java and implemented in tag handler classes. Hans Bergsten, whose book, *JavaServer Pages*, provides excellent information and instruction in JSPs, pushes this idea to the limit, effectively removing the need for Java within JSPs. However, it disturbs me to see the replacement of a relatively stable and well-known language (Java) with a new, less-known and less battle-tested language (his custom tag libraries).

If I were working at a large corporation that had decided to make a major investment in Java, servlets and JSPs, I would feel quite comfortable using custom actions. Such a company is in a position to create its own tag library

that can be used over the life of a web site, defining its own standards for how things work.

But for those of us working outside of a large corporation, or who work with a number of different clients, interoperability is a paramount concern. If each of my clients were to define a different set of custom actions for their sites, I would find myself struggling to remember which tags and attributes I need to use for loops, database access and conditional execution. And as I indicated above, I worry about working with nonprogrammers who already struggle with the idea of learning to embed Java inside of their HTML pages—teaching them two different types of loops (one in Java, and another with custom actions) will undoubtedly lead to some confusion.

A good compromise solution might be the inclusion of a large, standard set of custom actions that will be made part of the JSP specification, much as has been done with JavaBean-related tags. The tag library presented in Bergsten's *Java Server Pages* is a good start but is only one of many such available libraries. It would be nice to see the JSP community get together on this issue, before we find ourselves faced with dozens of similar but incompatible libraries, some of which will undoubtedly be proprietary.

Conclusion

JSPs are a powerful and quick way to work with server-side Java, particularly for nonprogrammers who don't want to learn a language. Custom actions, particularly when combined with JavaBean components, make it possible to perform complex tasks with a minimum of code. With some forethought, a site can avoid inserting nearly any Java code into their JSPs, relying instead on custom actions and tag libraries.

However, sites (and consultants who use custom actions) should balance the convenience and power of tag libraries with the fact that they are effectively creating a new programming language. If we aren't careful, custom tags will cause a split in the server-side Java community, fracturing it into subcommunities that use different, incompatible libraries.

Sidebar

Reuven M. Lerner owns and runs Lerner Communications Consulting, a firm specializing in web applications and internet technologies. He lives with his wife

and daughter in Modi'in, Israel. You can reach him at reuven@lerner.co.il or via the ATF home page, http://www.lerner.co.il/atf/.

Archive Index Issue Table of Contents

Advanced search

Programming Silence OUT!

Marcel Gagné

Issue #87, July 2001

Voice-recognition software: one step closer to HAL.

Is that not wonderful, François? Ever since I was a young boy, even before I thought of opening this restaurant, I have wanted something like this. I remember watching *2001: A Space Odyssey*, listening to the voice of HAL 9000 and thinking, "That is what I need. A talking computer." Years later, I suddenly realized that I still did not have my talking computer. Well, today, *mon ami*, we are going to change all that.

What? Our guests have arrived? Welcome, *mes amis*, to *Chez Marcel*. I am so happy you could come today. We have some wonderful items on our menu for the programmer who has programmed everything. Please, sit and François will bring you some wine. François, go to the cellar and fetch the 1996 Hill of Grace from Australia.

Get comfortable, *mes amis*. You are going to love this wine. Ah, *merci*, François. Please, pour for our guests.

I was telling François that we should have talking computers everywhere by now, but my Linux workstation spends its time in silence. For reasons that I cannot fathom, none of the software on my system seemed to be speech-enabled. So, for all of today's recipes, you will need a sound card in your system, properly configured, as well as a microphone.

The Center for Speech Technology Research (CSTR) at the University of Edinburgh in Scotland had just what I needed to start down the road to my own talking computer. By surfing over to this address, http://www.cstr.ed.ac.uk/projects/festival/, you'll find a fascinating project called Festival.

Festival is a multilingual speech synthesis system. It is capable of text-to-speech work with multiple voices. With its API design, it can be incorporated into

numerous other programs and applications. You'll see what I mean as we explore this package.

Getting Festival is easy. Visit the site, click the download button and pick up the latest version of the package. While you are there, pick up the **speech_tools** package, as Festival relies on its presence. Consequently, it is the first thing we will build:

```
tar -xzvf speech_tools-1.2.1.tar.gz
cd speech_tools
cp config/config-dist config/config
chmod +w config/config
```

At this point, you may want to consider whether you wish to use shared libraries since the default is not to do so. In that case, you must uncomment the following line in the config/config file by removing the hash mark before the word “SHARED”: **#SHARED=1**.

This is actually the recommended option now. Whether or not you choose to do so, we can now continue with the build:

```
make info
make
make install
```

This is an excellent time to relax, try the foie gras (it is really quite excellent, *non*?) and have another sip of your wine. When the speech_tools are built, it is time to create the Festival system. Unpack the source for Festival into a directory of your choosing with the command **tar -xzvf festival-1.4.1.tar.gz**.

All these files will expand into a directory called /festival. Before you do anything else, unpack the language lexicon and speech database. I started by grabbing the following files:

```
festlex_CMU.tar.gz
festlex_POSLEX.tar.gz
festvox_kallpc16k.tar.gz
```

The CMU file is a dictionary file for all English voices, while the POSLEX file contains speech parts also common for all English speakers. Finally, we have a speech database for an American-style voice, with 16k sampling. Different readers will no doubt want different voices, whether it be male, female, British or (*bien sûr*) French. Details on what you might need are available by looking at the README in the web site's distribution directory.

When you extract these files, they will also expand into the same /festival directory. From here, the process is identical to what we did for the speech_tools, right down to copying the config file from config-dist, except, of course, that we copy it to the /festival directory.

When this is all done, type **bin/festival** from the installation directory. You should see something like this:

```
bin/festival
Festival Speech Synthesis System 1.4.1:
release November 1999
Copyright (C) University of Edinburgh, 1996-1999.
All rights reserved.
For details type `(festival_warranty)'
festival>
```

Are you ready to hear your computer speak? Well then, at the festival> prompt, type the following (the parenthesis and quotes are important): **(SayText "Franiçois. Vite. More wine.")**.

If everything went well, you should have heard the words “François. Vite. More wine” come from the speakers. It is a commanding voice, *non*? I like to play that line because it unnerves my faithful waiter. Of course, since I am using an English voice and database, the pronunciation is what you might call interesting. Typing **control-d** here will let you exit the Festival command mode. You can also type **(quit)**. Once again, the brackets are important here. Let's try something more interesting. By using the --tts flag, we can specify the pathname to a text file and have Festival read it for us. For instance, I have a cron job that changes the message-of-the-day every night by running the fortune program. So, to read the message-of-the-day, I could do this:

```
bin/festival --tts /etc/motd
```

If you leave off a filename, you can just start typing. When you are done, you then press control-d to terminate the input, and Festival will exit. Here's one other thing to try. Simply pipe the output of a command to the Festival program. Want to hear a rather interesting interpretation of the date? Try this, **date | bin/festival --tts**.

You might also run the Fortune program for some synthesized wisdom: **/usr/games/fortune | bin/festival --tts**.

Festival can also run as a server for other programs to pass text information by running the program with the --server flag. As an example, you might write an application that writes to the Festival socket (by default, this runs on port 1314). Listing 1 is a little Perl script I wrote just for this occasion. It is not exciting, but you might consider it a starting point for your own applications. Keep in mind that you may have to change the path to your Perl executable in the first line.

Listing 1. Writing to the Festival Socket

A wonderful example of this idea is a program written by Darxus called **speechd**. This package implements a device file called /dev/speech to which you

can write any text you like. Redirect the output to this device, and the Festival system (when running in server mode) will pick it up and say it. This is also a Perl script and can be downloaded from the Speech IO site, http://www.SpeechIO.org/.

Start by unpacking the distribution into a temporary directory. Then, run the simple build that follows:

```
tar -xzvf speechd-0.54.tar.gz
cd speechd
make
make install
```

To run the program as a dæmon, simply type the path to the command: **/usr/local/bin/speechd**.

Before I continue, I should tell you that I had some problems here. My Red Hat system's /etc/hosts file had a localhost entry that read:

```
127.0.0.1   localhost.localdomain  localhost
```

Yours may as well.

To make things work properly, I had to change it so that the two localhost definitions were switched like this:

```
127.0.0.1   localhost localhost.localdomain
```

If, by chance, the Festival server is not running, the speechd will try to start it. Unfortunately, if you built Festival from source, you may have to modify the speechd script to use the full path to the executable. Another option is to copy the Festival binaries to /usr/local/bin.

So, what can you do with this? Well, using my original fortune program example, I could simply redirect the output to /dev/speech with the command **/usr/games/fortune > /dev/speech**.

Implementing this into your scripts is extremely easy. Here is another example. I could have a script that runs every few minutes to check for new mail, count the number of messages and tell me about it through the speech device. (Note that the single quotes are actually back-ticks.)

```
echo "You have `frm | wc -l` messages in your
mailbox" /dev/speech
```

Now that we have our Linux systems talking to us, it seems to me only one thing is missing. We need to have our systems listen to us and do as they are told, *non*? We need voice recognition software. For that little bit of

personalization, I went to Daniel Kiecza's home page and picked up the source for the latest **cvoicecontrol**, a nice little package distributed under the GPL.

With cvoicecontrol, we can start creating the fully automated system of our dreams. Of course, this now means we should be careful what we say, *non*?

```
tar -xzvf cvoicecontrol-0.9alpha.tar.gz
cd cvoicecontrol-0.9alpha
./configure
make
make install
```

The resultant files will appear in your /usr/local/bin directory. There are three that you need to know about. One is the cvoicecontrol program itself. Before you can start using it, you need to calibrate your microphone and create model files. This is done with the **microphone_config** and **model_editor** programs.

Start with the microphone_config program and follow it through the question and answer session. It is all nicely menu-driven. Your default mixer and audio devices should be automatically detected, so that part should already be filled in. Mine showed up as /dev/mixer and /dev/dsp. The next step is to adjust mixer levels, set up recording thresholds and create a configuration file. Probably the toughest part of this whole step is having to talk loud enough for the time it takes the program to get its levels. I tell you right now, *mes amis*, it is more difficult than it sounds. The default location is $HOME/.cvoicecontrol/config.

Then comes the fun part. Start the model_editor. You will be presented with a menu where you can load, edit, save or create a new speaker model. Have a look at Figure 1 to see the program in action.

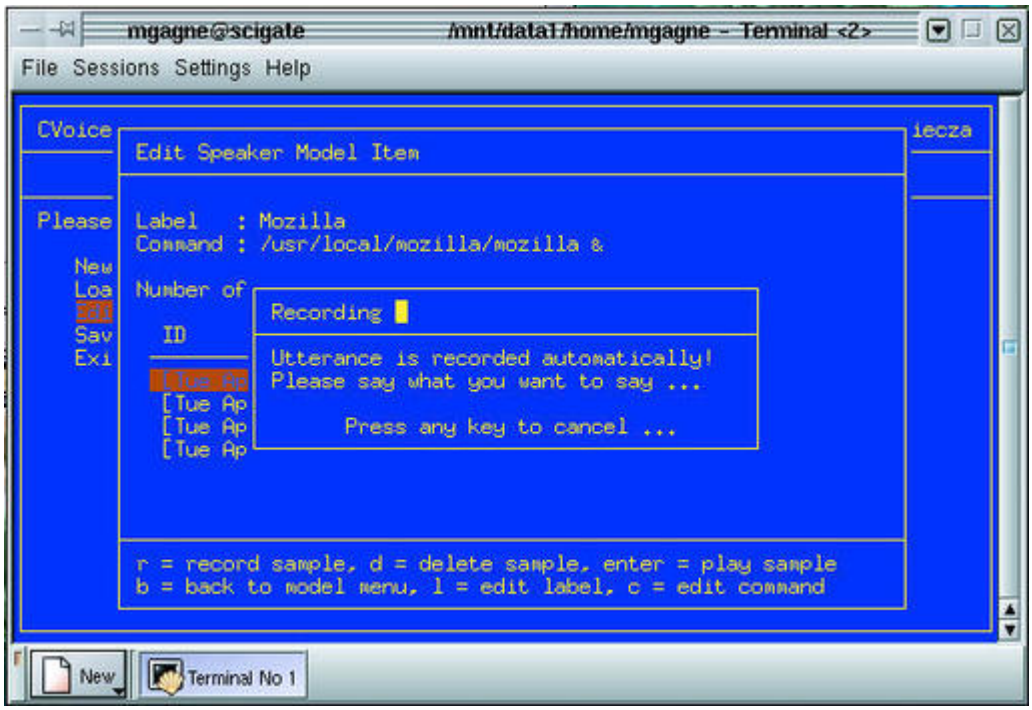


Figure 1. Adding a Command through the **model_editor**

The model is you, *mes amis*. Since there is nothing here yet, hit New Speaker Model, then choose Edit. Another menu will appear where you can record words and specify the events those sounds will generate. Everything here is a single keystroke. Press “a” to add a new item. The item will show up in the list as a generic item with no command specified. Now, press Enter to edit the item, and change the label to something that makes sense. For instance, I created one called “Start Mozilla”, and I entered **/usr/local/mozilla/mozilla &** for the Mozilla start command.

Notice that I put an ampersand at the end of the command to put it into the background. I did that because I want to be able to launch other voice commands after this one has started. Once you have done this, you need to enter some samples of your voice. You will need at least four samples. Speak your command clearly, wait and add another sample. When you have the four, you can back out (by pressing “b”) and save your speaker model. You may call it whatever you like, just remember where you put it.

So, now we have our microphone configured and at least one command associated with an equivalent voice command. By the way, to start Mozilla, you could just as easily say “browser” as your voice command, but it usually makes sense to use the command name, *non*? The only thing left to do is start cvoicecontrol: **cvoicecontrol speakermodel.cvc**.

Since I saved my speaker model as chefmarcel.svc (the extension is not necessary), I started the voice recognition software by typing **cvoicecontrol chefmarcel.svc**.

Now, if I say the word “Mozilla”, the browser starts up. I also created commands for my favorite editor, **vi**, and, of course, a couple of games.

Open the restaurant doors please, Tux. With tools such as the ones on today's menu, you can be well on your way to owning the computer of the future, today. You already run Linux, so you are almost there.

Well, *mes amis*, the clock, she is telling me time is running out and we must soon close. No sense in closing too soon, though. François, won't you please pour our guests another glass of wine? *Merci*, *mon ami*. You know, François, your built-in voice recognition software is working very well. Of course, François, I know you are a man and not a machine. Do not look so hurt. It is only a little joke, *non*?

Mes amis, I must thank you again for coming. Until next time, please join us here at *Chez Marcel*. Your table will be waiting.

A votre santé! Bon appétit!

Resources

Marcel Gagné lives in Mississauga, Ontario. In real life, he is president of Salmar Consulting Inc, a systems integration and network consulting firm. He is also a pilot, writes science fiction and fantasy, and is coeditor of TransVersions, a science fiction, fantasy and horror anthology. He loves Linux and all flavors of UNIX and will even admit it in public. He is the author of *Linux System Administration: A User's Guide*, coming soon from Addison Wesley. He can be reached via e-mail at mggagne@salmar.com. You can discover lots of other things from his web site at http://www.salmar.com/marcel/.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Intrusion Detection for the Masses

Mick Bauer

Issue #87, July 2001

Set up Tripwire to catch intruders big and small.

As impregnable as we hope our hardened systems are, security isn't a game of absolutes: the potential for system breaches must be recognized. Tripwire Open Source is a free and open-source software package that gives us a reasonable chance of being notified of possible breaches as soon as they occur.

Integrity checkers such as Tripwire create cryptographic “fingerprints” of system binaries, configuration files and other things likely to be tampered with in the course of, or subsequent to, a security breach. They then periodically check those files against the stored fingerprints and e-mail discrepancies back to you.

Tripwire is the most well known and mature integrity-checking system, and the one we're about to discuss in depth. You may also be interested in AIDE, which runs on more platforms than Tripwire Open Source, and FCheck, which is written 100% in Perl and, thus, even less platform-dependent than AIDE or Tripwire (it even runs on Windows). See the Resources section at the end of this article for links to AIDE's and FCheck's web sites.

Whither Integrity Checking?

Integrity checking mechanisms are like system backups; we hope we'll never need them, but heaven help us if we do and they're not there. Also, like system backups, integrity checking is an important component of a larger plan. If a system has been hardened, patched and maintained according to the industry's highest standards (or at least common sense), an integrity checker will provide a final safety net that helps minimize the damage done by whatever brilliant cracker manages to sneak in.

The principle on which integrity checkers operate is simple: if a file changes unexpectedly, there's a good chance it's been altered by an intruder. For

example, one of the first things a system cracker will often do after “rooting” a system is replace common system utilities such as **ls**, **ps** and **netstat** with “rootkit”, which makes them appear to work normally but conveniently fail to list files, processes and connections (respectively) that might betray the cracker's presence.

Integrity checkers can be used to create a database of hashes (checksums) of important system binaries, configuration files or anything else we don't expect or want to have changed. By periodically checking those files against the integrity checker's database, we can minimize the chances of our system being compromised without ever knowing it. The less time between a system's compromise and the administrator's learning of it, the greater the chance administrators can catch, or at least evict, the intruders.

One caveat: any integrity-checker with an untrustworty database is worthless. It is imperative to create this database as soon as possible after installing the host's operating system from trusted media. I repeat: installing, configuring and maintaining an integrity-checker is not worth the effort unless its database was initialized on a clean system.

Tripwire—the First and Still Foremost Integrity Checker

Among the most celebrated and useful things to come out of Purdue's COAST project (http://www.cerias.purdue.edu/coast/) is Tripwire, created by Dr. Eugene Spafford and Gene Kim. Originally both open source and free, Tripwire went commercial in 1997, and fee-free use was restricted to academic and other non-commercial settings.

Happily, last October Tripwire, Inc. released Tripwire Open Source, Linux Edition. Commercial versions of Tripwire until then had included features not available in the older Academic Source Release. In contrast, Tripwire Open Source is a more-or-less current version of the commercial product. Other than lacking enterprise features such as centralized management of multiple systems, it is very similar to the Tripwire for Servers product.

Note that Tripwire Open Source is free for use only on non-commercial Unices, i.e., Linux and Free/Net/OpenBSD. In fact, it's only officially supported on Red Hat Linux and FreeBSD, although there's no reason it shouldn't compile and run equally well on other Linux and BSD distributions. Only the older Academic Source Release is free for use on commercial Unices such as Sun Solaris and IBM AIX; the proprietary version must be purchased for these systems.

But we're all Linux geeks here, right? For the remainder of this discussion I'll focus on Tripwire Open Source, Linux Edition.

Obtaining, Compiling or Installing Tripwire

As of this writing, the most current version of Tripwire Open Source is 2.3.1-2. It can be downloaded as a source-code tarball at http://sourceforge.net/projects/tripwire/. I strongly recommend that you obtain, compile and install this version. While Tripwire has had only one significant security problem (and only a denial-of-service risk, at that) in its history, we use Tripwire because we're paranoid. For paranoiacs, only the latest (stable) version is good enough.

Having said that, the binary version included with Red Hat 7.0 is reasonably up-to-date. As far as I can tell, the differences between Red Hat's v2.3-55 RPM and the official source-release v2.3.1-2 involve non-security-related bugfixes; therefore you're probably taking no huge risk in using your stock RH 7.0 RPM. But don't say I told you to!

To compile Tripwire Open Source, move the archive to /usr/src and un-tar it, e.g., **tar -xzvf ./tripwire-2.3.1-2.tar.gz**. Next, check whether you have a symbolic link from /usr/bin/gmake to /usr/bin/make (non-Linux Unices don't all come with GNU **make**, so Tripwire explicitly looks for **gmake**--of course, on most Linux systems this is simply called make). If you don't have it, the command to create this link is **ln -s /usr/bin/make /usr/bin/gmake**.

Another thing to check for is a full set of subdirectories in /usr/share/man—Tripwire will need to place man pages in man4, man5 and man8. On my Debian system /usr/man/man4 was missing, and as a result the installer created a file called /usr/share/man/man4 that, of course, was actually a man page incorrectly copied to that name rather within it.

Finally, read the source's README and INSTALL files, change to the source-tree's src directory (e.g., /usr/src/tripwire-2.3.1-2/src), and make any changes you deem necessary to the variable-definitions in src/Makefile. Be sure to verify that the appropriate SYSPRE definition is uncommented (**SYSPRE = i386-pc-linux**, **SYSPRE = sparc-linux**, etc.).

Now we're ready to compile, type **make release**. This will take awhile, so now is a good time to grab a sandwich. When the build is done, navigate up one directory level, e.g., /usr/src/tripwire-2.3.1-2, and execute these two commands:

```
cp ./install/install.cfg .
cp ./install/install.sh .
```

Now open install.cfg with your favorite text editor; while the default paths are probably fine, you should at the very least examine the Mail Options section. This is where we initially tell Tripwire how to route its logs. If we set TWMAILMETHOD=SENDMAIL and specify a value for TWMAILPROGRAM,

Tripwire will use the specified local mailer (**sendmail** by default) to deliver its reports to a local user or group.

If instead we set TWMAILMETHOD=SMTP and specify values for TWSMTPHOST and TWSMTPPORT, Tripwire will mail its reports to an external e-mail address via the specified SMTP server and port. Note that if you change your mind later, Mail Options settings can be changed in Tripwire's configuration file at any time.

If the system on which you're installing Tripwire is a multiuser system, and one that you or other system administrators routinely log on to and read e-mail, the SENDMAIL method is probably preferable. If the system is a host you typically administer remotely from other systems, the SMTP method is probably better.

Once install.cfg is set to your liking, it's time to install Tripwire. Simply enter **sh ./install.sh**. You will be prompted for site and local passwords; the site password protects Tripwire's configuration and policy files, whereas the local password protects Tripwire databases and reports. This allows the use of a single policy across multiple hosts in such a way as to centralize control of Tripwire policies but distribute responsibility for database management and report generation.

A Note about RPMs

Needless to say, it's simpler and faster to install RPMs (but again, note that the most up-to-date version of Tripwire may not be available in this format). The only thing you need to know is that after you run **rpm**, you'll need to enter /etc/tripwire/**twinstall.sh** to generate site and local passwords. This script behaves much like the end of the source distribution's **install.sh** script—see the previous paragraph.

Using Tripwire

As useful as Tripwire is, it has a reputation for being difficult to configure (which is, of course, true of most powerful and flexible tools). But it's really not as bad as all that, and by following the simple instructions I'm about to set forth, you can use Tripwire effectively enough to catch yourself some bad guys. So now, let's enter the elite ranks of users who have not only installed, but actually used, Tripwire!

Managing the Configuration File

The first thing we need to do is double-check our configuration file, tw.cfg. Actually, this file was just encrypted by the installation script, but for our convenience a clear-text copy called twcfg.txt should reside in /etc/tripwire. This

is the place to change any settings you've had second thoughts about since running the installation script, and the variables are named accordingly.

If you make any changes, re-encrypt the configuration file with the command:

```
twadmin --create-cfgfile --site-keyfile ./site.key twcfg.txt
```

where site.key is the name of the site-key created at installation time and twcft.txt is the name of the clear-text configuration file you just edited and wish to encrypt. That may seem obvious given that these are the default names for these files, but you can name them whatever you like. Regardless, don't forget to specify the keyfile, or **twadmin** will return an error (remember, the point of this exercise is to encrypt the configuration file).

Warning: you should not, as a matter of practice, leave clear-text copies of your Tripwire configuration (tw.cfg) or policy (tw.pol) files on your hard drive. After editing and encrypting them, delete the clear-text versions. You can always retrieve them later with the command:

```
twadmin --print-cfgfile > mycfg.txt
```

where, predictably, you can substitute mycfg.txt with whatever you like.

Although I haven't yet described Tripwire binaries in any detail (it's more useful to explain them in context), you've no doubt guessed by now that twadmin is used to manage Tripwire's configuration, key and (initially, at least) policy files.

Managing the Policy File

Like the Tripwire configuration file, policies are edited as text files but are encrypted and signed before being installed. Unlike the configuration file, however, we only use the twadmin command to install a policy file for the first time on a given system; subsequently we'll use the **tripwire** command in policy-update mode.

In any event, the command to install a policy the first time after installing Tripwire is:

```
twadmin --create-polfile twpol.txt
```

where twpol.txt is the name of the clear-text policy file you wish to install.

As with configuration files, you shouldn't leave clear-text policy files on your system. If you need to refer to or edit the policy later, you can retrieve it by typing:

```
twadmin --print-polfile > mypol.txt
```

mypol.txt can be whatever you wish to call the clear-text copy of the policy. (See a pattern here?)

Editing or Creating a Policy

And now we begin the serious voodoo. Tripwire's policy file is its brain: it specifies what to look at, what to look for and what to do about it. It's also a little on the user-hostile side, though not nearly so bad in this regard as, say, sendmail.cf (but prepare to memorize some abbreviations).

Naturally Tripwire Open Source comes with a default policy file, and naturally you may, if you like, use this as your very own personal Tripwire policy. But since the default policy was created for a Red Hat system running nearly everything in the distribution, you should probably edit this policy heavily rather than use it as is.

First, a word about tuning. If your policy doesn't check enough files or doesn't look closely enough at the ones it does check, Tripwire's purpose is defeated and shenanigans will go undetected. Conversely, if the policy looks too critically at files you expect to change anyhow, Tripwire will generate “false positives” that serve no purpose other than to distract your attention from actual discrepancies.

It's doubtful you'll create a sane baseline the first time around. You'll almost certainly need to adjust your policy on an ongoing basis and especially after the first time you run an integrity-check. Thus, even if you do have a Red Hat system with exactly the same configuration as the one for which the default Tripwire Open Source policy was designed, you still need to learn proper Tripwire policy syntax. Let's get cracking.

I'm going to explain policy file structure and syntax by dissecting a working policy file into manageable chunks. The first chunk we'll look at is from the very beginning of our sample policy file and lists some variable definitions:

```
WEBROOT=/home/mick/www;
CGIBINS=/home/mick/www/cgi-bin;
TWPOL="/etc/tripwire";
TWDB="/var/lib/tripwire";
```

We can use these variables to save valuable touch-typing energy. On to the next chunk, some fancier variable definitions:

```
BINS          = $(ReadOnly) ; # Binaries that should
                                not change
SEC_INVARIANT = +tpug ;       # Dir.s that shouldn't
                                change perms/ownership
```

```
  SIG_MED       = 66 ;          # Important but not
                                  system-critical files
```

Unlike the first set of variable definitions that involved simple path-shortcuts, these are a bit fancier. The first line shows us how to set one variable to the value of another—similar to bash shell syntax, but note the parentheses around the second variable's name.

The second line defines a “property mask”; property masks are abbreviations of the file properties Tripwire examines. Since property mask strings can be cryptic and unwieldy, most people prefer to use variables to refer to them. In fact, Tripwire comes with a number of predeclared variables set to common property masks, and the first line actually refers to one of these, ReadOnly, a property mask for files that shouldn't change in any way, like binaries. We'll discuss property masks in detail, but all in good time.

The third line creates a name for a severity level. Severity levels can be used to differentiate between rules of varying importance. When the tripwire command is invoked with the --severity N parameter, only rules with assigned severity levels equal to or greater than N will be parsed. If this parameter is not used, all rules will be parsed. Also note that if a rule has no severity level associated with it, the level will be set to zero by default. That is, that rule will only be parsed when the --severity parameter isn't specified.

Now that we've got a feel for policy variables and what they're used for, let's start looking at actual rules:

```
  # Mick's Web Junk
  (
    rulename = "MickWeb",
    severity = $(SIG_MED),
    emailto = mick@uselesswebjunk.com
  )
  {
    $(WEBROOT)              -> $(ReadOnly) (recurse=1) ;
    !$(WEBROOT)/guestbook.html ;
    $(CGIBINS)              -> $(BINS)     ;
    /var/log/httpd          -> $(Growing)  ;
  }
```

This is a very rich chunk, so we'll begin with rule structure. Rules may either stand alone or be grouped together based on common attributes; this listing shows a group of rules (contained within curly brackets) with several shared attributes (in parentheses, above the rules). This group's rulename is “MickWeb”, the group's severity is 66 and reports involving this group will be e-mailed to mick@uselesswebjunk.com. Note that attributes are separated by commas, whereas each rule ends with a semicolon.

Attributes can also be assigned to individual rules. The first rule in this group has the attribute recurse set to 1, which means that the directory /home/mick/

www will be checked down one level (i.e., the directory itself plus everything immediately “below”, but no further). Note that by default, directories will be recursed as far down as they go; in effect, the recurse attribute has a default value of “True”.

Attributes listed in rule statements usually override those listed in parentheses above such rules' group. The exception is the attribute “emailto”, which is cumulative: if a group has a shared emailto string and one of that group's rules has a different emailto string, reports relevant to that rule will be e-mailed to all e-mail addresses in those two strings.

By the way, there are only four attributes: rulename, severity, emailto and recurse. For more detailed information on these see the Resources section.

After the group attributes for MickWeb we have some actual rules. Note the use of variables to specify both objects (the Tripwire term for files and directories) and property masks. In fact, none of the rules uses a “longhand” property mask. This is common practice and perfectly acceptable.

Immediately below the first rule, which tells Tripwire to treat the first level of my WWW directory as read-only, we have a statement beginning with an exclamation point. This statement is called a stop point, and it defines an exception to a rule. In this case, the stop point tells Tripwire to ignore changes to the file /home/mick/www/guestbook.html. Attributes do not apply to (nor may they be assigned to) stop points.

There, that's a complete policy file (technically, at least—it doesn't check any system binaries or configuration files at all—real policies are much longer). Listing 1 shows it in all its non-dissected glory.

Listing 1. Sample Policy File

You may have noticed this entire file only contains one explicit reference to a property mask: the variable declaration in which SEC_INVARIANT is set to “+tpug”. What does that mean?

A property mask is a series of file/directory properties that should be checked or ignored for a given object. Properties following a plus sign are checked; those following a minus sign are ignored. The properties are abbreviated as outlined in Table 1.

Table 1. Property Mask Values

Tripwire's own documentation describes these properties in depth. If you're unfamiliar with some of the more arcane file attributes (e.g., inode reference

count) I recommend the paper “Design and Implementation of the Second Extended Filesystem” by Card, Ts'o and Tweedie (see Resources, below). As for hash-types, note that you generally won't want to use more than one or two cryptographic hashes per rule: these are CPU-intensive. On the other hand, do not rely solely on CRC-32 hashes, which are fast but much easier to subvert.

As I mentioned earlier, Tripwire has a number of predefined (hard coded) variables that describe property masks, shown in Table 2.

Table 2. Predefined Tripwire Property Mask Variables

In most cases it's simpler to use these predefined masks than to “roll your own”. Note, however, that you can combine these variables with additional properties, e.g.,

```
/dev/console  -> $(Dynamic) -u ;
# Dynamic, but UID can change
```

is the same thing as

```
/dev/console  -> +pingutd-srlbamcCMSH-u
```

After you've created what seems like a reasonable policy, you need to install it. Again, the command to install a system's first Tripwire policy is:

```
twadmin --create-polfile policyfile.txt
```

The last step in setting up Tripwire is to create (initialize) its database. Important: there's no point in initializing a Tripwire database on a system that's been up and, therefore, has possibly been compromised already! Tripwire installation, configuration and initialization should occur as soon after OS installation as possible.

To initialize the database, we now use the tripwire command: **tripwire --init**. Doesn't get much simpler than that, does it? But use the --init directive only when creating a new database. If you need to change your Tripwire policy later, it's better to use the following commands:

```
twadmin --print-polfile > mypolicy.txt
  # dump current installed plcy
vi mypolicy.txt
  # make changes to policy
tripwire --update-policy mypolicy.txt
  # install new policy --
  # DON'T USE TWADMIN FOR THIS!
```

Running Checks with Tripwire

Once we've got a database installed, we can run periodic checks against it. At its simplest, the command to do so is: **tripwire --check**. This compares all protected

files against the hash-database and prints a report both to the screen and a binary file. The report can be viewed again with the command:

```
twprint --print-report --report-level N --twrfile /path/file
```

where N is a number from 0 to 4, 0 being a one-line summary and 4 being a full report with full details. /path/file is the full path and name of the latest report (by default it should reside in /var/lib/tripwire/report).

To have Tripwire automatically e-mail the report to all recipients specified in the policy, we could have run our check like this instead:

```
tripwire --check --email-report
```

Note that the report is still printed to standard output and saved in /var/lib/tripwire/report as well.

If you've installed the Tripwire RPM on a Red Hat 7 system, your system is already set up to run Tripwire periodically in check mode. The RPM includes the script /etc/cron.daily/**tripwire-check**. If you've used the emailto attribute in your Tripwire policy, however, you may wish to edit the second-to-last line of this script to read:

```
test -f /etc/tripwire/tw.cfg && /usr/sbin/tripwire --check --email-report
```

(This line by default lacks the --email-report flag.)

Tripwire won't tell you much unless you run regular checks, either manually, via cron/anacron or some combination thereof.

There Were Violations! Now What?

So, what happens when Tripwire reports violations? That's up to you. Often, violations will be the result of a too-restrictive Tripwire policy rather than actual skullduggery. You'll need to decide which are which and what to do about them.

Either way, you'll probably want to update the Tripwire database after violations are found so that it reflects any legitimate changes to the files and directories being monitored. There are two ways to do this. The first is to run the tripwire command in update mode:

```
tripwire --update --twrfile /var/lib/tripwire/report/myhost-date.twr
```

The last argument is the absolute path to the report you wish to use as the basis for this update. This opens the report with the editor specified in tw.cfg so

you can indicate which, if any, of the changed files/directories you wish Tripwire not to update in its database. In other words, when you exit the editing session, Tripwire will update the attributes and hashes in its database only for those report entries with an X next to them (they all are by default).

Here's an excerpt from a tripwire-update session:

```
Remove the "x" from the adjacent box to prevent
updating the database with the new values for this
object.
Modified:
[x] "/home/mick/www"
```

If I delete the “x” from this entry, exit the editor and run a check, the /home/mick/www change will be reported again; the database will not have updated to reflect this change. In short, if the change is legitimate, leave the “x” there. If it isn't or you're not sure, remove the “x”.

The second way to update the Tripwire database is to do the actual check in “interactive” mode, which immediately triggers an update session after the check finishes. Thus,

```
tripwire --check --interactive
```

is the same thing as

```
tripwire --check
tripwire --twrfile /var/lib/tripwire/report/reportname.twr
```

but with the added advantage of saving you the trouble of looking up the report's filename (since it includes a timestamp, this isn't easily guessed).

When you get false positives, it will often make sense to fine-tune your policy. Remember to do this in the manner described at the end of the Editing or Creating a Policy section above.

Now Go Forth and Trip Yourself Some Crackers!

Before we sign off for this month, I leave you with two excellent tips I learned from Ron Forrester, Project Manager for Tripwire Open Source:

1. Always set MAILNOVIOLATIONS=TRUE [in tw.cfg] so you get a heartbeat from tripwire, i.e., if your cron job runs Tripwire once an hour, and you don't get a report for more than an hour, you know something is up.
2. Always leave a violation or two (say /etc/sendmail.st) in—this makes it more difficult for an intruder to forge a report. It is quite easy to forge a report with no violations, but add a known violation or two, and it gets much more difficult.

I hope that's enough to get you started; there's much I haven't covered here or have only touched on. Believe you me, this tool's power is worth its learning curve, and the *Tripwire Open Source Manual* (see below) is both comprehensive and extremely well-written. Good luck!

Resources

Sidebar

Mick Bauer (mick@visi.com) is a network security consultant in the Twin Cities area. He's been a Linux devotee since 1995 and an OpenBSD zealot since 1997, taking particular pleasure in getting these cutting-edge operating systems to run on obsolete junk. Mick welcomes questions, comments and greetings.

Archive Index Issue Table of Contents

Advanced search

Linux at NAB

Robin Rowe

Issue #87, July 2001

Linux goes to the movies.

This month we're taking a break from installing Linux video software to report on the cool Linux video applications shown at the National Association of Broadcasters (NAB) convention in Las Vegas, April 21-26, 2001. Every April, television chief engineers, station managers and everyone else with an interest in professional video gear make the trek to Las Vegas for the NAB. With more than 115,000 attendees and 1,700 exhibits, NAB is one of the largest tradeshows in the world. Computer enthusiasts may be more familiar with CES (122,000 attendees and 1,000 exhibitors) and COMDEX (225,000 attendees and 2,300 exhibitors). By acreage all three shows are about the same size: a million square feet. Bring your hiking boots.

At NAB Linux was a much bigger presence than last year. Special effects offerings included RAYZ from Silicon Grail, Shake from Nothing Real and Maya from Alias|Wavefront. Linux Media Arts showcased the Cinelerra Quicktime video editor and the Kino DV editor. AMD briefed video developers on getting the most from Athlon. BOXX Technologies offered the *3D*BOXX Graphics workstation. Video board makers presented Hauppauge and DVS. For Linux settop boxes, ATI, OpenTV and Phillips TiVo were there. Khronos and ProMpeg Forum each made announcements of significant Linux media APIs. And, at the NAB conference, there was a presentation on Linux and MP3 by Radio Free Asia.

“The vast majority of high-end production shops will convert to Linux in the next eighteen months”, says Ray Feeney, a four-time Academy Award-winner for Scientific and Engineering Achievement and the technical chair of the Visual Effects Society (http://www.visual-effects-society.org/). VES participants include special effects artists from ILM, DreamWorks, Pixar, PDI, Disney's The Secret Lab and Sony. VES members are discussing how to transition the industry to Linux. For years SGI IRIX was the OS of choice for studio film projects, and many

haven't been satisfied with the support they have received since switching to Windows. Because Linux is open source, it is an OS for which the industry can develop motion picture features, and its ubiquity makes it much easier to find users more familiar with it than with IRIX.

“Nobody is buying anything now”, said Feeney at NAB, “but once the writers' strike situation is over there will be restructuring. Today when people have rendering requirements they are going to Linux. The rendering side is solved and turnkey.” Feeney compares the state of the art in Linux graphics workstations to where Linux servers were two years ago when first used in a major production, for rendering *Titanic*. He expects Linux will dominate in two years on graphics workstations, not just servers. ILM has already begun using Linux for workstations as well as rendering.

Feeney is also the founder of Silicon Grail, a company showing their next generation compositing tool RAYZ in the Compaq booth at NAB. Designed for creating feature film effects, RAYZ runs on Linux, IRIX and Windows. Production that originated on film is first digitized by a film scanner such as the Kodak Cineon. Although each frame may be stored as a separate image file (e.g., in Targa format), to the RAYZ operator it still looks like a movie. Effects filters are diagramed left-to-right in the graph window for ease of manipulation, and the effects results are visible in the video playback window. Completed movies may be transferred back to film or to video tape.

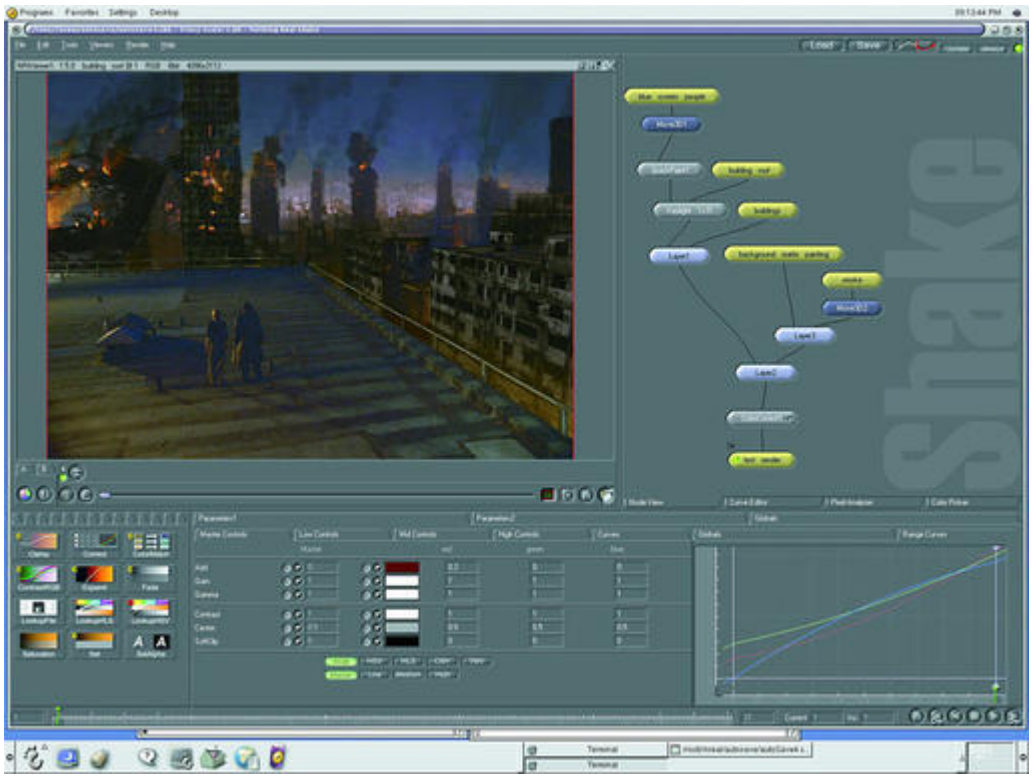
RAYZ is Silicon Grail's next generation interactive compositor based on their Chalice software. Chalice was used on *Deep Blue Sea*, *Prince of Egypt*, *Star Trek: Insurrection*, *Titanic*, *Fantasia 2000*, *Men in Black* and many other motion pictures. It provides effects and color correction tools. For a typical configuration RAYZ is expected to cost $9,900. RAYZ was in public beta at the time of NAB, for release in May (see http://www.silicongrail.com/).

Maya Linux Screenshot

Maya from SGI company Alias|Wavefront is a widely used professional 3-D animation and visual effects package. At NAB, Alias|Wavefront announced Maya 4, with enhancements and optimizations in the areas of rendering, character animation, brush and paint tools, and games-related functionality. Enhancements to the Maya non-linear motion editing technology include time warping, character merging, drag and drop, and character set editing. New character animation features include switching between forward kinematics and inversion kinematics, quaternion-based IK, motion trails, ghosting and a jiggle deformer to wobble character muscles.

“Customer demand for a Linux version of Maya has driven this development”, says Alias|Wavefront entertainment business general manager Bob Bennett. That decision was influenced by feedback from the Technical Committee of the Visual Effects Society. Linus Torvalds called Maya 3 “the most complex and powerful 3-D graphics application ever to run on Linux” when the Linux port was announced in the summer of 2000.

Maya was used by ILM for *The Perfect Storm*, by The Secret Lab for *Mission to Mars*, by Sony Pictures Imageworks for *The Hollow Man* and by many other productions. Alias|Wavefront demonstrated Maya 4 for IRIX and Windows at NAB, due for release the end of June. The Linux version is expected six to eight weeks later. Prices start at $7,500 (see http://www.aliaswavefront.com/).



Shake 2.4 Linux Screenshot

Shake from Nothing Real (Venice, California) is high-speed compositing software optimized for visual effects for feature films. New features in Shake 2.4 include vector-based procedural paint, advanced color correction tools, a new rotoscoping node and ease-of-use improvements. It is resolution independent and automatically handles different bit-depths (8, 16, 32) and resolutions (Web, 601, HDTV, film, IMAX).

Shake has been used in over 60 feature films, and since its debut, has been used in every Oscar winner for visual effects including *Titanic*, *What Dreams May Come*, *The Matrix* and *Gladiator*. Shake runs on Linux, IRIX and Windows. Shake 2.4 entered beta testing in February and is scheduled for release shortly after the NAB. Prices start at $9,900 (see http://www.nothingreal.com/).

Linux Media Arts (Burbank, California) creates turnkey video editing and media-streaming systems for video, film, audio and the Internet. LMA president Mike Collins says, “Our goal is to make Linux the premier multimedia editing and media production platform in the world, largely using open-source software.” To that end Collins says their mission right now is to create servers and editors for a new high-quality SDDI board they announced at NAB. Until now they have been offering M-JPEG and DV systems.

At NAB LMA demonstrated their DV and Quicktime-based editing systems. Bundled software includes Cinelerra and Kino video editors, Blender 3D, the GIMP, Corel Draw and Red Hat Linux. Systems are based on AMD, Intel and

Compaq Alpha chips. Prices start at $1,395 with an AMD 1GHz processor (see http://www.linuxmediaarts.com/).

Kino is a simple cuts-only DV video editor. DV is the format used by most consumer digital camcorders. Author Arne Schirmacher in Germany says, “Kino allows you to record, create, save, edit and play movies recorded with DV camcorders. Although it has windows and menus, it is actually a keyboard-driven program. It uses many keyboard commands that are similar to the vi text editor.” (See www.schirmacher.de/arne/kino/0).

At NAB Jason Howard of LMA demonstrated Kino 0.4 as part of a suite of tools for DV to DVD production. Howard is the developer of **dvcont**, a DV camera-control utility. Other open-source DV utilities include **dvgrab** (also by Schirmacher) and **dvsend**. The dvgrab utility moves footage from the camera to the computer using the 1394 protocol. The dvsend utility moves edited material back onto camcorder video tape.

Howard is also a partner in the video production company Spectsoft (Oakdale, California). Currently a mixed Windows/Linux facility, partner Ramona Howard (Jason Howard's mom) is keen to move entirely to Linux in order to have increased control over the production process. As a video producer she encourages Jason to develop more open-source software. Jason Howard says, “All it takes is one person to have a passion in Linux. That's all of us, and a lot of soda!”

Howard is currently working on a transcoder utility to move footage between AVI and Quicktime to enable transferring material easily between Kino and Cinelerra. He is also working on selectively grabbing DV camera footage based on an edit decision list (see http://www.spectsoft.com/).

Cinelerra is a Quicktime-based video editor from Heroine Virtual. They are well known in the Linux community for their Broadcast 2000 editor (see “Moviemaking in a Linux Box?” January 2001) and XMovie player. Cinelerra was announced and shown at NAB. It is a resolution-independent editor with over 100 real-time video and audio filters, has support for five-channel audio and is designed to run on multiprocessor hardware. Heroine Virtual bundles its software with systems manufactured by Linux Media Arts (see http://heroines.sourceforge.net/).

Chipmaker AMD (Sunnyvale, California) briefed video developers at NAB on how to take advantage of processor advances in their Athlon and Duron chips. Audio/video development engineer Jim Bovenzi says, “Dual processor solutions are key to maximizing performance in video applications. Dual Athlons provide incredible performance.” AMD demonstrated the OnAir DTV 1080i HDTV card

from Sasem running on Windows XP (see http://www.amd.com/ and www.sasem.com).

WinTV-HD is an HDTV card offered by Hauppauge (Hauppauge, New York). The Hauppauge WinTV card is very popular among Linux users. Unfortunately, according to a Hauppauge engineer the WinTV-HD architecture is quite different. The Linux community faces another reverse engineering job to support the WinTV-HD card shown at NAB. Hauppauge has no plans for Linux support. The WinTV-HD card costs $399 (see http://www.hauppauge.com/). Another Windows consumer HDTV card at NAB was accessDTV for $479 (see http://www.accessdtv.com/).

The HDStationOEM is an uncompressed HDTV real-time I/O card for professional applications. Maker DVS Digital Video (Glendale, California) says they offer Linux drivers. However, the $40,000 price tag puts it squarely in the pro market. In supports 1080p and even 2k film resolution (see http://www.digitalvideosystems.com/).

The Linux-powered TiVo settop box sold by Phillips was on display, as was a new settop from ATI Technologies (Ontario, Canada) based on their ALL-IN-WONDER. An engineering model running on a Linux PC was in their booth. ATI offers Linux drivers for their new Fire GL4 high-end graphics card that's $1,995. Settop developer OpenTV (Mountain View, California) announced their intention to support Linux (see www.tivo.com, www.ati.com and www.opentv.com/).

3D

*3D*BOXX is a series of high-performance Windows and Linux workstations offered by BOXX Technologies (Austin, Texas) for digital applications such as film, HDTV, video and game development. One user of *3D*BOXX Linux machines is Blur, a visual effects, animation and design studio located in Venice, California. They produced a four-minute stereo 3-D 70mm ride for Paramount Parks called The 7th Portal. *3D*BOXX systems start at $2,309. *Render*BOXX server systems start at $2,692 (see http://www.boxxtech.com/).

OpenML is a new API that hopes to do for video software developers what OpenGL has done for graphics developers. The Khronos Group consists of leading graphics and digital media companies including 3Dlabs, ATI, Discreet,

Evans & Sutherland, Intel, NVIDIA, SGI and Sun Microsystems. The Khronos Group announced at NAB that they had completed the OpenML 1.0 Specification. OpenML is based on the dmSDK, which SGI recently made open source, and MLdc, an abstraction layer for display devices. Video effects software maker Discreet is helping to drive the requirements. Neil Trevett of 3Dlabs who made the announcement says, “We hope to encourage an open-source implementation in Linux.” (See http://www.khronos.org/ and http://oss.sgi.com/.)

The release of the AAF (Advanced Authoring Format) software development kit version 1.0 was announced by the Pro-MPEG Forum at NAB. The software is intended to interchange video, audio and metadata in postproduction applications (see http://www.aafassociation.org/ and http://sourceforge.net/projects/aaf/).

Besides exhibitors, there was a Linux-specific session in the NAB conference: “Linux and MP3 for Archiving”. Radio Free Asia (www.rfa.org) broadcasts via shortwave radio and the Web in Tibetan, Cantonese, Uyghur, Burmese, Vietnamese, Lao, Khmer (to Cambodia) and Korean (to North Korea). RFA is a private corporation funded by the US Congress to bring news and information to populations lacking a free press. Their first broadcast was to China in September 1996. RFA broadcasts 34 hours of programming per day in nine languages.

RFA production support manager A. J. Janitschek and lead technical engineer Tom Hallewell described the evolution of the RFA MP3 archive technology. “At Radio Free Asia we used to archive all of our audio as WAV files onto 4GB DAT tapes”, said Janitschek. But that was expensive, labor intensive and inconvenient to search and retrieve. “With our current system we can store 100 hours of archive programming on a single CD-ROM and make it available to anyone who has an MP3 player on their PC”, added Hallewell.

RFA uses about 15 CD-ROMs to archive a month's audio (1,020 hours). They use three different 16-bit encoding settings: 1) 32khz, 48kbs and 20MB/hr for broadcast quality (over shortwave); 2) 16khz, 16kbs and 7MB/hr for long-term storage; and 3) 12khz, 14kbs and 6MB/hr for broadcast on the Web.

RFA doesn't encode their MP3 streams on a PC, but uses the Telos Audioactive Realtime MPEG Internet Audio Encoder. This rackmount box multicasts MP3s that they capture on their Linux archive system and burn onto CD-ROMs. Janitschek says the advantages of a hardware encoder include real-time performance and the fact that MP3 patent licenses (to Fraunhofer and THOMSON) are covered. The unit sells for $2,800 (see http://www.audioactive.com/).

RFA not only uses Linux to archive their content, they also are contributing to open-source software. R-BOSS (Radio-Broadcast Open Source System) is a collection of digital broadcast content management applications written using Python. Also available is their 3D-Project, a free distribution of broadcast specific 3-D drawings, material and texture bitmap files. For further information or to download, see http://www.techweb.rfa.org/.

Radio Free Asia Screenshot

And there were other Linux exhibitors at NAB. Real Networks offered streaming and playing on Linux. Kasenna was showing their asset management system, which supports Linux. Thomcast Communications introduced the DCX Millennium Digital Transmitter, which can be controlled by Linux. No doubt there were many more Linux applications that we missed.

Next month we will return to our project of upgrading our Debian Linux installation to the 2.4 kernel and XFree86 4.x GUI. Time permitting, we will also install an ATI All-In-Wonder Radeon graphics card.

Robin Rowe is a partner in MovieEditor.com, a technology company that creates internet and broadcast video applications. He has written for *Dr. Dobb's Journal*, the *C++ Report*, the *C/C++ Users Journal* and *Data Based Advisor*. His software designs include a client/server video editing system in use at a Manhattan 24-hour broadcast television news station, Time Warner New York

One and associated web site http://www.ny1.com/. You can reach him at robin.rowe@movieeditor.com.

Archive Index Issue Table of Contents

Advanced search

Integrating a Linux Cluster into a Production High-Performance Computing Environment

Troy Baer

Issue #87, July 2001

Troy discusses the performance and usage of the Brain at the Ohio Supercomputer Center.

In August 1999, the Ohio Supercomputer Center (OSC) entered into an agreement with SGI, in which OSC would purchase a cluster of 33 SGI 1400L systems (running Linux). These systems were to be connected with Myricom's Myrinet high-speed network and used as a “Beowulf cluster on steroids”. The plan was to make this cluster system eventually a production quality high-performance computing (HPC) system, as well as a testbed for cluster software development by researchers at OSC, SGI, Myricom and elsewhere.

OSC was no stranger to clustering, having built its first workstation cluster (the Beakers, eight DEC Alpha workstations running OSF/1 and connected by FDDI) in 1995. Also, the LAM implementation of MPI started at OSC and was housed there for a number of years. This was not even OSC's first Linux cluster; Pinky, a small cluster of five dual-processor Pentium II systems connected with Myrinet, had been built in early 1999 and was made available to OSC users on a limited basis. However, this new cluster system was different in that it would be expected to be a production HPC system, just as OSC's Cray and SGI systems were.

Hardware Configuration

The new cluster, nicknamed the Brain (after Pinky's smarter half on Animaniacs), consisted of 33 SGI 1400L systems, each with four Pentium III Xeon processors at 550MHz, 2GB of memory, a 10/100Mbps Ethernet interface and an 18GB UW-SCSI system disk. One system was configured as a front end or interactive node with more disks, a second Ethernet interface and an 800Mbps high-performance parallel interface (HIPPI) network interface. The

other 32 systems were configured as compute nodes, with two 1.28Gbps Myrinet interfaces each. The reason for putting two Myrinet cards in each system was to increase the available network bandwidth between the nodes; the SGI 1400 systems have two 33MHz 32-bit PCI buses, so one Myrinet card was installed in each PCI bus (a single Myrinet card can easily saturate a 33MHz 32-bit PCI bus, so installing two in a single PCI bus is not a good idea). The 64 Myrinet cards were initially connected to a complex arrangement of eight 16-port Myrinet switches designed to maximize bisection bandwidth (the amount of bandwidth available if half of the network ports simultaneously attempt to communicate with the other half), but in the final installation these were replaced with a single 64-port Myrinet CLOS-64 switch. A 48-port Cisco Ethernet switch was also purchased to connect to the Ethernet cards in each system. This Ethernet network is private; the only network interface to the cluster accessible from the outside is the second Ethernet interface on the front-end node.

It may seem like overkill to have three separate types of networks (Ethernet, Myrinet and HIPPI) in the cluster, but there is actually a good reason for each. Ethernet is used mainly for system management tasks using TCP/IP protocols. HIPPI is used on the front end for high-bandwidth access to mass storage (more on this later). Myrinet, on the other hand, is intended for use by parallel applications using the MPI message-passing library. For the Brain cluster (as well as its predecessor, Pinky), the MPI implementation used was MPICH, from Argonne National Laboratory. The reason for selecting MPICH over LAM was that the developers at Myricom had developed a ch_gm driver for MPICH that talked directly to the GM kernel driver for the Myrinet cards, bypassing the Linux TCP/IP stack entirely and allowing for much higher bandwidth and lower latency than would be possible over TCP/IP. There have been several other MPI implementations for Myrinet, such as FM (fast messaging) and AM (active messaging), but these did not appear to be as robust or well supported as MPICH/ch_gm.

Installation

The system was initially assembled and tested in one of SGI's HPC systems labs in Mountain View, California during October 1999. It was then shipped to Portland, Oregon where it was featured and demoed prominently in SGI's booth at the Supercomputing '99 conference. After SC99, the cluster was dismantled and shipped to OSC's facility in Columbus, Ohio where it was permanently installed.

The final installation bears some discussion with respect to floor space, power and cooling. As finally installed, the cluster was comprised of seven racks, six with five 1400 nodes each and one with three 1400 nodes, the Myrinet CLOS-64 switch, the Ethernet switch and a console server (see Figure 1). One of SGI's on-

site computer engineers (CEs) estimates that each rack weighs something on the order of 700 pounds, and he insisted on having the raised floor in the area where the cluster was installed reinforced (to put this in perspective, the only other OSC system that required floor reinforcement was a Cray T94, which weighs about 3,800 pounds). Each SGI 1400 unit has three redundant power supplies rated at 400 watts, requiring a total of twenty 20-amp circuits to be installed to supply electrical power. The front-end node was placed on UPS, while the compute nodes were placed on building power. Cooling for the room was found to be adequate; the heat load generated by 33 1400Ls ended up being inconsequential next to the cooling requirements for OSC's Cray systems and the Ohio State University's mainframes, all of which are housed in the same facility.

Figure 1. The Cluster as Finally Installed

Interface with Mass Storage

OSC's other HPC systems at the time of the Brain cluster's installation consisted of the following:

- mss: an SGI Origin 2000 with eight MIPS R12000 processors at 300MHz, 4GB of memory, 1TB of Fibre Channel RAID, and approximately 60TB of tapes in an IBM 3494 tape robot with four tape drives
- origin: an SGI Origin 2000 with 32 MIPS R12000 processors at 300MHz and 16GB of memory
- osca: a Cray T94 with four custom vector processors at 450MHz and 1GB of memory

- oscb: a Cray SV1 with 16 custom vector processors at 300MHz and 16GB of memory
- t3e: a Cray T3E-600/LC with 136 Alpha EV5 processors at 300MHz and 16GB of memory

The latter four systems all mounted their user home directories from mss using NFS over a HIPPI network. When the cluster was installed, its front-end node, known as oscbw or node00, was added to the HIPPI network (see Figure 2). In addition, to make staging files into the compute nodes easier for end users, the compute nodes were configured to mount the user home directories over the private Ethernet, using a previously unused Ethernet port on mss (see Figure 3).

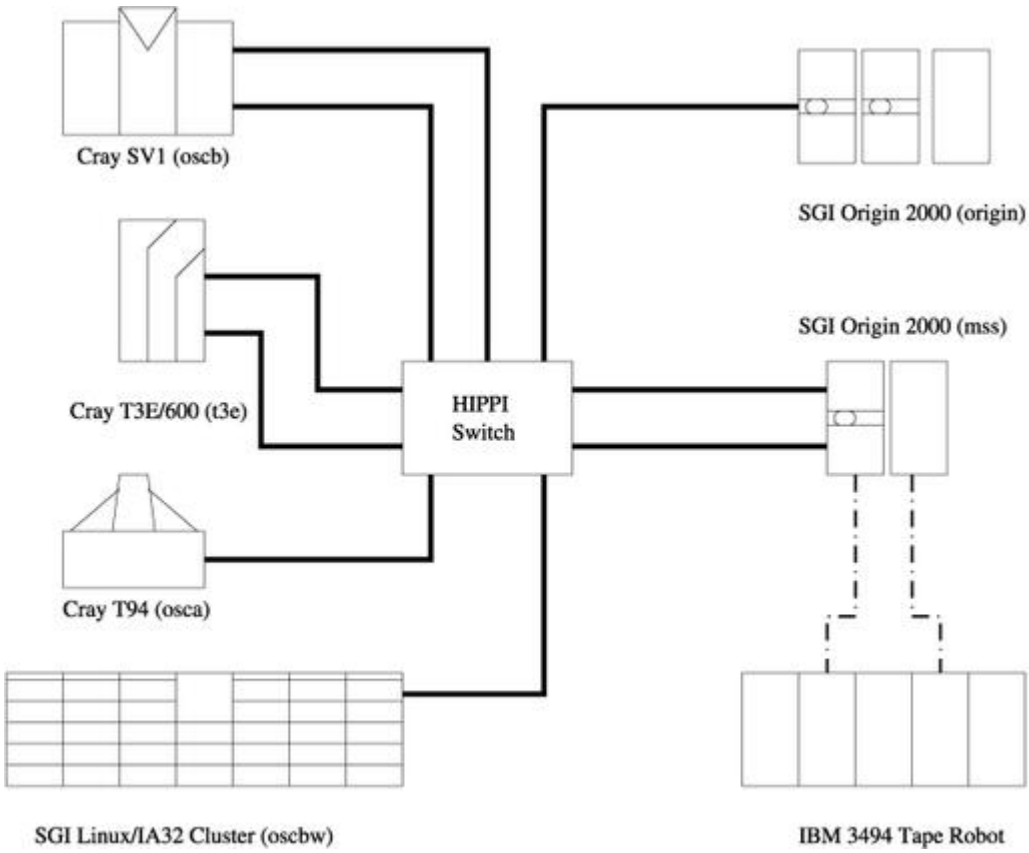


Figure 2. HIPPI Network

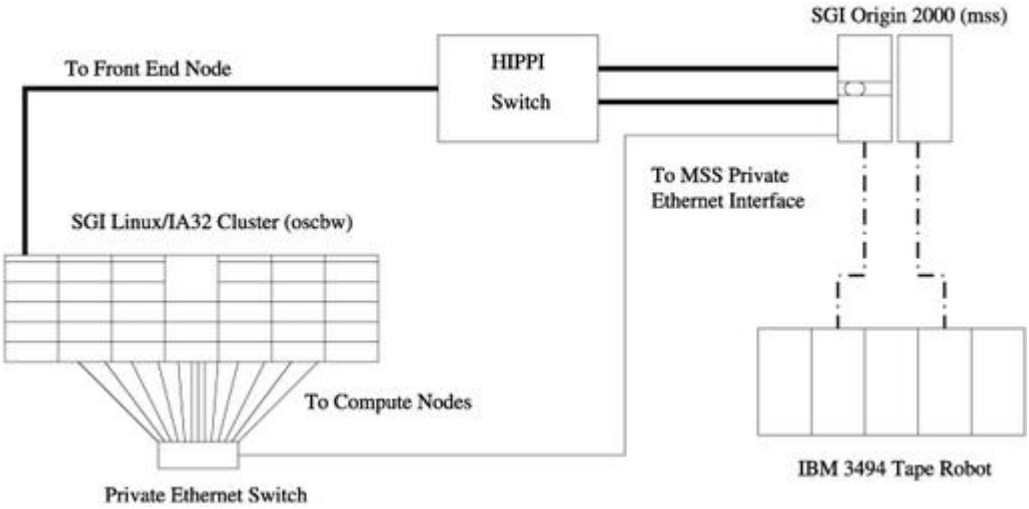


Figure 3. Compute Nodes Networked with mss via Private Ethernet

One difficulty encountered with this arrangement involved an interaction between the Linux NFS client implementation and hierarchical storage management (HSM). The mss system runs an HSM product from SGI called Tape Migration Facility (TMF). TMF periodically scans though all the files stored on selected filesystems (in this case the users' home directories) looking for large files that have not been accessed in some time and thus can be migrated off to a tape in the 3494 robot. When a user attempts to read a file that has been migrated to tape, the initial read() system call blocks until TMF is able to migrate the contents of the file back to disk. Unfortunately, the Linux 2.2 NFS client implementation queued NFS file reads by NFS server rather than by filesystem, and so trying to read a migrated file often caused the front-end node to lock up while the file was retrieved from disk.

Job Scheduling and Accounting

As with OSC's other HPC systems, the Brain cluster represents a shared resource for researchers at various academic and industrial institutions in Ohio. The Portable Batch System (PBS) version 2.2 was selected to handle resource management and job scheduling on the cluster. This choice was based on several factors:

- Previous experience: PBS version 2.0 had been used on the Pinky cluster, after an extensive comparison with Platform Computing's LSF suite.
- Use at large sites: many large Linux cluster sites used PBS as their scheduling software, including the National Aerodynamic Simulation (NAS) facility at NASA Ames where PBS had been developed.
- Source availability: PBS was an open-source product with considerable Linux support, whereas LSF was closed source and Platform showed little interest at the time in making all of LSF's features available under Linux.

- Cost: PBS was freely available (although support contracts were available from MRJ), while LSF incurred a significant per-processor licensing cost for production use.

Version 2.2 of PBS had another feature that was a significant improvement over version 2.0: per-processor allocation of cluster nodes. In version 2.0, PBS classified a system as either a time-shared host (e.g., a Cray vector system or large SMP) that could multitask several jobs or a space-shared cluster node (e.g., a uniprocessor node in a Beowulf cluster or IBM SP) that could be allocated to only a single job. PBS 2.2 extended the cluster node concept with a "virtual processor" attribute; a cluster node with multiple virtual processors can have multiple jobs assigned to it, and a user can specifically request nodes with multiple virtual processors per node.

However, PBS required some tinkering to make it work the way OSC's administrators and users had come to expect from a batch system after ten years of using NQE on Cray system. First, each job was assigned a unique working directory (accessed through the $TMPDIR environment variable). PBS job prologue and epilogue scripts were written to create these directories at the start of the job and delete them at the end of the job (see Listings 1 and 2). Scripts were also added to the /etc/profile.d directory on each compute node to set $TMPDIR inside batch jobs (see Listings 3 and 4). A distributed copy command, **pbsdcp**, was developed to allow users to copy files to $TMPDIR on each of the nodes allocated to their job without needing to know a priori which nodes they would be given (see Listing 5).

Listing 1. PBS Job Prologue Script

Listing 2. PBS Job Epilogue Script

Listing 3. Script to Set $TMPDIR Inside Batch Jobs

Listing 4. Script to Set $TMPDIR Inside Batch Jobs

Listing 5. pbsdcp

To facilitate the use of graphical programs such as the Totalview parallel debugger, a mechanism for doing remote X display from within the cluster's private network was developed. This mechanism relied on the X display port forwarding feature of ssh, as well as the interactive batch job feature of PBS. An interactive job in PBS is just like a normal batch job, except that it runs an interactive shell instead of a shell script. With an X pseudo-display on the front-end node courtesy of ssh, it was possible to make X programs run on the cluster's private network using some unorthodox **xauth** manipulations (see Listings 6 and 7).

Listing 6. Manipulating xauth to Display X Programs on the Private Network

Listing 7. Manipulating xauth to Display X Programs on the Private Network

The target applications for the Brain cluster were MPI-based parallel programs. To improve the startup time of these programs and make their CPU time accounting accurate, the rsh-based **mpirun** shell script from MPICH was replaced with a C program called **mpiexec**, which uses the task management API in PBS to start the MPI processes on individual nodes. This program also allowed a user to specify the number of MPI processes with which their job was run as one per virtual processor, one per Myrinet interface or one per node (see Listing 8 for examples). The OSC mpiexec program is available under the GNU GPL (see Resources for details).

Listing 8. Manipulating mpiexec

Users are charged against their time allocations based on the number of processors used and the duration of use. In the case of the Brain cluster where resources are space-shared, charging is done by multiplying the wall clock time used by a job times the number of processors requested. PBS supplied accounting logs with records of wall clock time and processors used, which were processed by a short Perl program and inserted into OSC's user accounting database. (The reader should keep in mind that no money changes hands for academic use of OSC's systems; researchers are simply granted time based on peer review of their research proposals.)

User Environment

Great care was taken to make sure that the interactive environment on the cluster was as friendly and as similar to the other OSC systems as possible. The nodes of the cluster mount their home directories from mss, just as the other systems do. OSC's Cray systems use a modules facility for dynamically modifying environment variables to point to different versions of compilers, libraries and other software; the cluster nodes were given a workalike facility, originally developed at Los Alamos National Laboratory. Also, a complete suite of compilers for C, C++, Fortran 77 and Fortran 90, as well as a debugger and profiling tool, were purchased from the Portland Group. These compilers were selected based on their excellent optimizer, which was originally developed for the Pentium Pro-based ASCI Red TFLOPS system at Sandia National Laboratory. A variety of numerical libraries were made available on the cluster, including both open source (FFTw, PETSc and ScaLAPACK) and closed source (NAG Fortran and C).

One area in which the Brain cluster is rather unique is parallel performance analysis. Performance analysis tools under Linux have historically been rather

primitive compared to those available on real supercomputers such as the Cray T3E. However, OSC staff were able either to acquire or develop a respectable collection of performance analysis tools for the Brain cluster. For profiling of serial (i.e., nonparallel) code, both the GNU **gprof** command-line profiler and the Portland Group's **pgprof** graphical profiler were installed. For profiling of MPI-based parallel code, the MPICH distribution supplied a profile logging facility and a Java-based graphical analysis tool called **jumpshot**. Finally, for truly in-depth performance analysis, the author developed an analysis program for hardware performance counter data called **lperfex**.

Hardware performance counters are a feature built into most modern microprocessors, and Intel processors based on the P6 core (i.e., Pentium Pro, Pentium II, Pentium III, Celeron and Xeon processors) have them as part of the model-specific registers. Erik Hendriks, one of the original Beowulf programmers and now at Scyld Computing, developed a kernel patch and user-space library for accessing these counters. The lperfex used this library to make a command-line performance counter interface, based on the example of the **perfex** utility found on SGI Origin 2000 systems. The beauty of this tool is that it requires no special compilation; it simply runs another program and records performance counter data (see Listing 9 for an example). It can also be used with MPI parallel applications (see Listing 10 at ftp://ftp.linuxjournal.com/pub/lj/listings/issue87/). As with the OSC mpiexec program, lperfex is available under the GNU GPL. Recent versions of the PAPI instrumentation library from the Parallel Tools Consortium have also been shipped with lperfex as part of the distribution.

Listing 9. An Example of the lperfex Counter Tool at Work

User Experiences

The Brain cluster was opened to friendly users in February 2000 and quickly gained a small but loyal following in the OSC user community. Somewhat to the chagrin of the OSC staff, not all of these users were interested in running parallel MPI applications. Many were interested in running older computational chemistry codes such as Amber and Gaussian 98, neither of which support MPI over Myrinet. Another rather novel application run on the cluster was a gene sequence analysis tool called NCBI BLAST, which Dr. Bo Yuan (a researcher in the Human Cancer Genomics Program at Ohio State University's college of medicine) used to annotate about sixty thousand genes from the draft version of the human genome data set in about one week's time. While not written with MPI, BLAST did run in parallel with four processors on a node by using pthreads and shared memory, and further concurrency was achieved by running multiple simultaneous jobs with each analyzing a different sequence. The pattern-matching algorithm used by BLAST is primarily integer arithmetic, and

the Intel processors in the Brain cluster's nodes were found to outperform the MIPS processors in OSC's Origin 2000 systems significantly (see Figure 4).

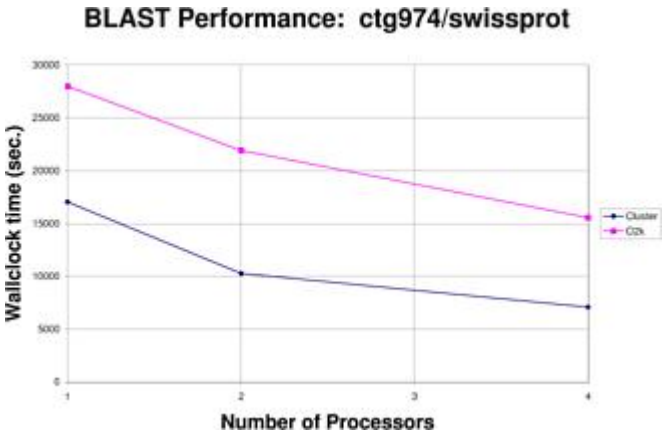


Figure 4. BLAST Performance

One user application that did use MPI over the Brain cluster's Myrinet network was a quantum chromodynamics (QCD) code written by Dr. Greg Kilcup from Ohio State's physics department. This code simulates the interaction of quarks in subatomic particles and is very communication-intensive, with each process sending a small message approximately every 200 floating point operations. This application is very sensitive to MPI latency and available memory bandwidth. On the Brain cluster, MPI latency was quite acceptable (on the order of 13 microseconds), and memory bandwidth became the main performance bottleneck. With four processors sharing an 800MB/s peak pipe to memory, each processor was limited to about 150MB/s sustained memory bandwidth. This limited each processor's floating point performance to about 60 MFLOPS. Using two processors per node improved both the sustained memory bandwidth and the floating point performance per processor (see Figure 5), while allowing higher processor-count runs than using only one processor per node.

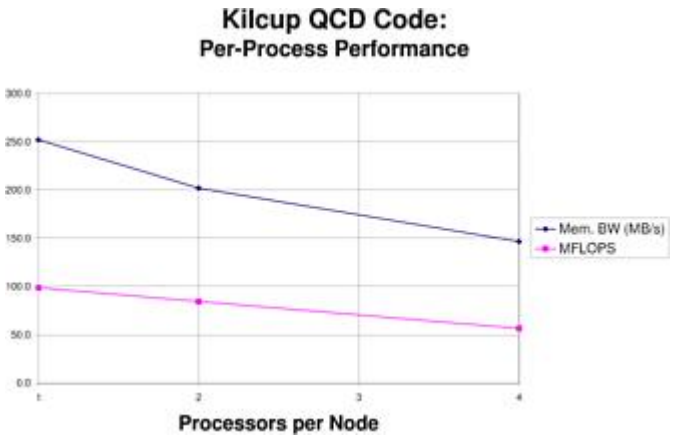


Figure 5. Performance Using Two Processors per Node

Another user code that used MPI over Myrinet on the Brain cluster was a Monte Carlo simulation of condensed matter physics, written by Dr. Mark Jarrell from

the University of Cincinnati's physics department. This application is “pleasantly” (also known as “embarrassingly”) parallel, meaning that it performs very little communication. However, like the QCD code described above, this code was very sensitive to memory bandwidth. The innermost loop of this application performed an outer product of two large (1,000+ element) arrays. This tended to cause low L2 cache reuse, which increased pressure on the already limited saturated memory bus (see Figure 6). As with the QCD code, this application has a sweet spot of two processors per node. Work is ongoing at OSC to try to improve the performance of this application.

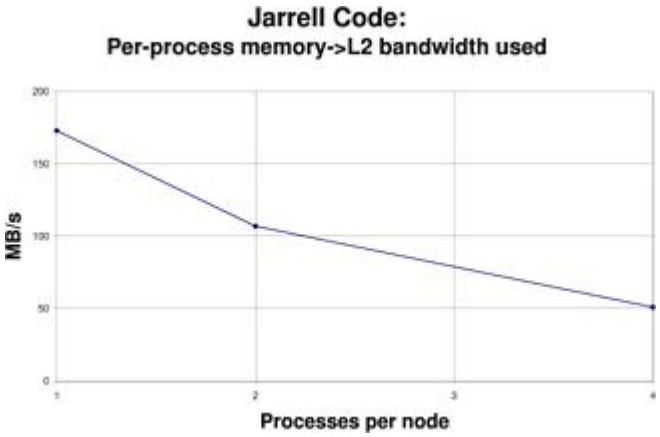


Figure 6. Jarrell Code Performance

Outstanding Issues and Future Directions

Overall, OSC has been quite pleased with the two Linux clusters it has had so far, and Linux clusters are seen at the center as one of the main directions in the future of high-performance computing. However, there are numerous areas in which Linux could be improved to support high-performance computing. Probably the most critical of these from OSC's perspective is a parallel filesystem with better parallel performance than NFS. The main use for this would be temporary storage for jobs; this is currently handled on the Brain cluster by having a $TMPDIR directory on each node, but a globally accessible scratch area would be much easier on users. There are currently two potential open-source candidates for a cluster parallel filesystem under Linux: GFS, from the University of Minnesota and PVFS, from Clemson University (see “A Parallel Virtual Filesystem for Linux Clusters”, *LJ* December 2000). GFS is a journaled, serverless storage-area network (SAN) filesystem over Fibre Channel. It promises to be an excellent performer, and its serverless aspect is quite attractive. However, as of this writing, the GFS code is in a state of flux following a redesign of its locking mechanisms, and the Fibre Channel switches needed for large topologies remain relatively expensive. PVFS, on the other hand, requires no special hardware; it, in effect, implements RAID-0 (striping) across multiple I/O node systems. PVFS's main downside is that it currently has no support for data redudancy, such that if an I/O node fails the parallel filesystem may be corrupted.

Another area where open-source solutions for high-performance computing clusters may be improved is job scheduling and resource management. While PBS has proven to be an adequate framework for resource management, its default scheduling algorithm leaves much to be desired. Luckily PBS was designed to allow third-party schedulers to be plugged into PBS to allow sites to implement their own scheduling policies. One such third-party scheduler is the Maui Scheduler from the Maui High Performance Computing Center. OSC has recently implemented Maui Scheduler on top of PBS and found it to be a dramatic improvement over the default PBS scheduler in terms of job turnaround time and system utilization. However, the documentation for Maui Scheduler is currently a little rough, although Dave Jackson, Maui's principal author, has been quite responsive with our questions.

A third area for work on Linux for high-performance computing is process checkpoint and restart. On Cray systems, the state of a running process can be written to disk and then used to restart the process after a reboot. A similar facility for Linux clusters would be a godsend to cluster administrators; however, for cluster systems using a network like Myrinet, it is quite difficult to implement due to the amount of state information stored in both the MPI implementation and the network hardware itself. Process checkpointing and migration for Linux is supported by a number of software packages such as Condor, from the University of Wisconsin, and MOSIX, from the Hebrew University of Jerusalem (see “MOSIX: a Cluster Load-Balancing Solution for Linux”, *LJ* May 2001); however, neither of these currently support the checkpointing of an arbitrary MPI process that uses a Myrinet network.

The major question for the future of clustering at OSC is what hardware platform will be used. To date Intel IA32-based systems have been used, primarily due to the wealth of software available. However, both Intel's IA64 and Compaq's Alpha 21264 promise greatly improved floating point performance over IA32. OSC has been experimenting with both IA64 and Alpha hardware, and the current plan is to install a cluster of dual processor SGI Itanium/IA64 systems connected with Myrinet 2000 some time in early 2001. This leads to another question: what do you do with old cluster hardware when they are retired? In the case of the Brain cluster, the plan is to hold a grant competition among research faculty in Ohio to select a number of labs that will receive smaller clusters of nodes from Brain. This would include both the hardware and the software environment, on the condition that idle cycles be usable by other researchers. OSC is also developing a statewide licensing program for commercial clustering software such as Totalview and the Portland Group compilers, to make cluster computing more ubiquitous in the state of Ohio.

Acknowledgements

This article would not have been possible without help from the author's coworkers who have worked on the OSC Linux clustering project, both past and present: Jim Giuliani, Dave Heisterberg, Doug Johnson and Pete Wyckoff. Doug deserves special mention, as both Pinky and Brain have been his babies in terms of both architecture and administration.

Troy Baer (troy@osc.edu) has been a systems engineer at the Ohio Supercomputer Center since January 1998. He holds bachelor's and master's degrees in aeronautical and astronautical engineering from Ohio State University. He has been a Linux user since first encountering it at NASA Lewis (now Glenn) Research Center in 1993. In his copious free time, he enjoys reading books and playing electric guitar.

Archive Index Issue Table of Contents

Advanced search

Whose Hand Is That in Your Pocket?

Doc Searls

Issue #87, July 2001

Sometimes too much just isn't enough. --Dean Landsman

Back when Steve Jobs was still at NeXT, he was interviewed by Robert X. Cringely for a PBS special called “Triumph of the Nerds”, a televised version of Cringely's brilliant book *Accidental Empires: How the Boys of Silicon Valley Make Their Millions, Battle Foreign Competition, and Still Can't Get a Date*. The best moment in the show came when Cringely asked Jobs what he thought about Microsoft.

Jobs leaned back, put on his best ironic smile and said, “They have no taste.” There, in four perfect one-syllable words, Jobs not only nailed Microsoft, but himself as well. True: while Microsoft has no taste, Jobs has nothing but.

Tastelessness has hurt Microsoft about as much as it has McDonalds. That much is obvious. Less obvious is what gets camouflaged by taste-free product names and dull-as-dirt descriptions of every new “enabling service”. Take .NET for example. It was announced a year ago, and still nobody can say what it is. When it came out, serious geeks like Joel Spolsky called it camouflage. “Read the white paper closely”, he wrote, “and you'll see that for all the hoopla, .NET is just a thin cloud of FUD. There's no there there. Try as you might to grasp onto something, the entire white paper does not say anything. The harder you grasp, the more it slips right through your fingers.”

HailStorm was different. While the .NET announcement was anesthesia, HailStorm was a wake-up call—at least for the geeks. The feature that really set them off was an innocent-sounding something called Passport. Here's Joel again:

> Am I the only one who is terrified about Microsoft Passport? It seems to me like a fairly blatant attempt to build the world's largest, richest consumer database, and then make fabulous profits mining it.

> It's a terrifying threat to everyone's personal privacy and it will make today's “cookies” seem positively tame by comparison. The scariest thing is that Microsoft is advertising Passport as if it were a benefit to consumers, and people seem to be falling for it!

Let's pause to visit two well-camouflaged reasons why Microsoft's products seem to achieve ubiquity so easily.

First is where the company comes from. Since the beginning (in 1975), Microsoft has been about personal computing. They care about users first and everything else after that. The “Micro” in their name isn't accidental.

Second, they pay very close attention to what users say they want. I would bet that what one user says to one tech support person has a far better chance of influencing software engineering at Microsoft than at any other software company. In fact, one high-up guy at Microsoft once told me that some of the company's software is full of features that “exactly one person asked for”.

Like every other gigantic industrial company that makes products for millions of individuals, Microsoft runs into problems when it begins to consider those individuals as something less than customers. That something is the name that shows up in the midst of Passport's URL: www.passport.com/Consumer/default.asp. That's right, you're a consumer. You're what Jerry Michalski calls a “gullet”: a creature that hangs around under the far end of the supply chain's conveyor belt, where you live only to “gulp down products and crap out cash”.

Pathetic creatures, consumers. Here's the HailStorm white paper, “Building the User-centric Experience” on the awfulness poor consumers must suffer with the PalmOS:

> If you want to enter a friend's new phone number into your PC, you use a keyboard and a piece of software like Microsoft Outlook to do it using a particular sequence of keystrokes and mouse clicks. But to enter that same information into your Palm Pilot, you need to learn a completely new interface—right down to relearning how to draw the letters of the alphabet! This environment, in which users are forced to adapt to technology instead of technology adapting to users, creates significant restrictions on how effective any application or web site can be....

And what would that awful environment be? Try a *market*. But who wants to live in a *real* market? Too messy. Too noisy. Too thick with too many vendors, and too many customers figuring things out, asking annoying questions. Too much like a real world. What the consumer wants is cocoon-like habitat like the

one they get with TV: a supply system of one-way channels wired up like the vast battery-charger in *The Matrix*.

From the matrix end, HailStorm is a set of “services”. From the battery's end, it's the Passport:

> A HailStorm-enabled device or application will, with your consent, connect to the appropriate HailStorm services automatically. Because the myriad of applications and devices in your life will be connected to a common set of information that you control, you'll be able to securely share information between those different technologies, as well as with other people and services....
>
> The HailStorm architecture is designed for consistency across services and seamless extensibility. It provides common identity, messaging, naming, navigation, security, role mapping, data modeling, metering, and error handling across all HailStorm services. HailStorm looks and feels like a dynamic, partitioned, schematized XML store. It is accessed via XML message interfaces (XMIs), where service interfaces are exposed as standard SOAP messages, arguments and return values are XML, and all services support HTTP Post as message transfer protocol.

Of course this all comes at a price:

> Microsoft will operate the HailStorm services as a business. The HailStorm services will have real operational costs, and rather than risk compromising the user-centric model by having someone such as advertisers pay for these services, the people receiving the value—the end users—will be the primary source of revenue to Microsoft. HailStorm will help move the Internet to end-user subscriptions, where users pay for value received.

In other words, Microsoft will turn the commercial side of the Internet into a big on-line service. But not one on the cable TV model, where you pay for a spigot at the end of the pipe. Oh, no, you want an á la carte model, where you, the battery, pays for everything. This is much more fair and efficient than the current credit-card system, which takes a cut out of the sell side of the final transaction.

Remember when Microsoft tried to buy Intuit? The banking and credit card industries went bonkers and lobbied successfully against the deal. HailStorm cuts them out almost entirely, but I suspect they have no idea what's really going on here. The camouflage is working.

But not with developers. Microsoft can't do this alone. So here's what's in it for the small fry:

> Microsoft will also derive some revenue from developers to help cover the costs of the services and products they need....
>
> Service operators will also have a certificate-based license relationship with Microsoft....That certificate will make it possible to filter abusers out of the system. Obtaining a certificate and the ongoing right to use HailStorm services will have a cost associated with it.

So there we have it: a blueprint for internet-based commerce, built by a Microsoft-enabled software industry, intermediated over Microsoft infrastructure, built by Microsoft and its allied developers.

I want to pause here to note that I'm not just bashing Microsoft here. I'm bashing what we've always called “consumerism” but which we would more accurately call producerism—the belief that Production can tell Consumption what it wants. A couple months ago in this space I gave Dell and Gateway some grief for doing the same thing with the cattle-chute choices it forced on visitors to their web sites.

With Hailstorm, Microsoft extends the cattle chute system to one by which every gullet can plug directly into the end of every value chain. Given the ubiquity of Windows, that's exactly what will happen. Or that's the idea.

The problem here isn't that Hailstorm cuts out competitors, or intermediaries, or anybody else. It's that it cuts out markets. It bypasses the bazaar. It embraces and extends what producerism has succeeded in doing ever since John Wanamaker invented the price tag in the late 1800s.

The challenge for the rest of us is to do with markets what we did with the Net in the first place: create ubiquitous conditions that make matrix-building impossible. That requires something many of us are not accustomed to doing: thinking about commerce. Specifically, thinking about markets as places, as environments, rather than as targets for stuff shoved down through the industrial distribution matrix.

For cultural reasons alone, this won't be easy. In *Homesteading the Noosphere*, Eric S. Raymond said that, among the “varieties of hacker ideology” there is a certain “hostility to commercial software and/or the companies perceived to dominate the commercial software market.” But hackers also built the Net, which was such an obviously promising environment for business that investors spent something like a trillion dollars funding fantasies about it,

putting the technology sector on a cocaine binge from which it will take years to recover.

Obviously, most of the investors didn't know what the hackers were really up to. Here's Eric S. Raymond, in early '99, way before the bubble burst: “We hackers were actively aiming to create new kinds of conversations outside of traditional institutions. This wasn't an accidental byproduct of doing neat techie stuff; it was an explicit goal for many of us as far back as the 1970s. We intended this revolution.”

With Linux we built a bazaar for developers. Now it's time to build one for everybody else. Let's create an interstructure for commerce which, like the Net, nobody owns, everybody can use, and anybody can improve.

Real markets are public places. You can't privatize what only works because it's public. But if we don't have something that works for everybody, somebody's going to build it for themselves. And it won't be pretty.

Doc Searls is senior editor of *Linux Journal* and coauthor of *The Cluetrain Manifesto*.

Archive Index Issue Table of Contents

Advanced search

Linux at the Embedded Systems Conference

Rick Lehrbaum

Issue #87, July 2001

Embedded Linux has really arrived!

In my writings about past embedded systems conferences, I've characterized the impact of Linux on the embedded market as a “disruptive technology” that was rewriting the rules of the game. Not any more—the disruption now appears to be complete!

Not that this was a big surprise, for a recently released subscriber survey conducted by *Embedded Systems Programming* magazine already exposed embedded Linux as the number two embedded OS in terms of “consideration” for new designs and number four in terms of actual usage.

The “big three” embedded OSes are, at the moment: 1) VxWorks, 2) Embedded Linux and 3) Windows Embedded—or 1) VxWorks, 2) Windows Embedded and 3) Embedded Linux—depending on how you count. These days, I doubt if you can find an embedded software or hardware vendor that doesn't attribute “must-support” status to Linux, or a developer who doesn't feel compelled to investigate Linux as an embedded OS option in the course of a new product development cycle.

In fact, last week's *ESC Official Show Guide* had, for the first time, an entire product category devoted specifically to “Embedded Linux” (there wasn't one for Windows Embedded or VxWorks, by the way). That category included 40 companies who weren't, by any means, all the companies at the conference that showed or promoted embedded Linux-oriented products. And, that doesn't include seven companies that exhibited in the Embedded Linux Consortium's “Pavilion 101”: RidgeRun, Tuxia, FS FORTH-SYSTEME, *Embedded Linux Journal*, Trolltech, PalmPalm and Wipro who were not allowed to have their own company listings in the show guide.

So that's the good news. The bad news, on the other hand, is that it's no longer feasible for me to cover “all things Linux” in my customary review of Linux at the Embedded Systems Conference. My sincere apologies, in advance, to all those companies and products that have not been included!

Most of the “traditional” (if 12-18 months qualifies as “traditional”) embedded and real-time Linux players were present, plus a few newcomers. These included distribution providers, tools vendors, purveyors of middleware (browsers, GUIs, protocol stacks, etc.), chip makers and board/system manufacturers. Following is a rundown on some of the many embedded Linux-related products and demos that I found at ESC.

Altera announced availability of an enhanced Nios soft processor core that will enable Ethernet/internet connectivity and will be well supported by embedded Linux through a strategic relationship with Microtronix.

Applied Data Systems (ADS) had their usual highly integrated, graphics-oriented, StrongARM-based single-board computers, such as the Graphics Master and Bitsy, running embedded Linux along with Century Software's Microwindows.

Century Software occupied a prominent position in Red Hat's large booth, where they showed several demonstrations of their well known Microwindows GUI/windowing environment for embedded devices and handheld computers. While there, I stumbled upon a preliminary data sheet for a new and not-yet-announced product called WebMedia, described as: “a collection of development tools, SDKs, runtime utilities (including embedded browser and specialized plugins), and applications that work together to form a very powerful, interactive, user interface framework...for settop boxes and web tablets.”

Grammar Engine announced that the next version of the PromICE memory emulator debugging tool is going to be based on NetSilicon's NET+ARM system-on-chip processor running uClinux Embedded Linux. As a result, open-source firmware residing in the tool will be available for modification and customization by the developers who use it.

Green Hills demonstrated a prototype of their “Multi” debugger running embedded Linux on a PowerPC target processor. Green Hills expects to release this support sometime this summer, for x86 and PowerPC targets.

Insignia demonstrated their newly announced Jeode platform for embedded Linux-based devices, a PersonalJava compatible implementation that includes Java-AWT compatible graphics. Currently, the AWT support requires a full X

Window System, but small alternatives such as Tiny-X and Micro-X will be investigated soon.

Lineo was spread around in four different booths. Among other things, they demonstrated their newly announced Board Development Kits (BDKs), for third-party single-board computers and microprocessor reference platforms, and showcased their “board farm”, which allows web-based access by developers to shared development platforms in Lineo's labs. A demonstration of the Embedix “Target Wizard” showed how easy it is to configure a highly customized, minimum footprint embedded Linux system. Lineo's recently acquired Convergence Integrated Media showed off their “Linux TV”, an open solution for digital TV. Lineo announced the opening of a new “embedded systems center” in Silicon Valley, availability of a new timing and schedulability tool for Embedix from Tri-Pacific Software, a Metrowerks CodeWarrior Development System for Embedix on the Motorola PowerQUICC II MPC8260 and an expanded partnership with Trolltech centered around Qt/Embedded support for Embedix. Lineo also had several science fair-like demonstrations including a laser modem and an RTAI-based software radio (see photo).

LynuxWorks announced BlueCat Linux 3.1, which adds support for MIPS R3000 and R4000 processors. With this release, LynuxWorks now claims to offer “the broadest microprocessor support for embedded Linux”. Also announced was Metrowerks' CodeWarrior IDE support for BlueCat Linux system development on Linux and Solaris hosts and the integration of Qt/Embedded into BlueCat Linux.

Microtronix showed off their newly announced Linux Development Kit for Altera's Nios core, which will be added to Altera's Nios Development Kit, an FPGA development platform that uses the Nios soft core embedded processor. In support of this effort, Microtronix ported uClinux to the Nios processor.

MontaVista unveiled Hard Hat Linux Version 2.0, which now includes a menu-based system builder tool (called a Target Configuration Tool), a utility to shrink shared libraries by eliminating unnecessary code and symbols and introduces embedded Linux support for Hitachi SH-3 and SH-4 microprocessors. MontaVista also announced they are releasing their CompactPCI hot swap technology source code to the Open Source community and declared their intention to integrate Trolltech's Qt/Embedded with Hard Hat Linux. Their many interesting demonstrations of Hard Hat Linux (HHL) included ones showing: HHL high availability; HHL cross development using multiple targets (PowerPC, x86, StrongARM, XScale, MIPS and SH) and graphical IDEs, debuggers and performance analysis tools; HHL along with IBM's VAME Java VM used as the basis of an automobile console; an iPAQ PDA running HHL with a Qt/

Embedded-based GUI; and HHL running on reference boards for Intel's XScale processor, Alchemy's Au1000 system-on-chip and Hitachi's SH-4.

OnCore demonstrated Linux for Real-Time and the OnCore OS, which provides the capability to run multiple OSes (including one or more copies of Linux) simultaneously on a single computer system. OnCore showed off their recently announced ability to emulate Wind River's VxWorks (and run VxWorks applications without modification) as one of the hosted OSes and demonstrated operation of their OS on the IBM PowerPC 405GP.

PalmPalm demonstrated their Tynux Embedded Linux in a number of gadgets including an iPAQ, a cell phone/PDA and a PDA/cell phone. The latter two are covered briefly below in the section on demos (including photos).

Rappore demonstrated their recently announced Bluetooth stack for embedded Linux-based devices. They are developing a sockets-based wireless API that supports the intermixing of both 802.11 and Bluetooth connectivity so that multiple devices can communicate freely in a mixed-technology environment, without regard to standard.

Red Hat had two interesting customer applications, an Intel Residential Router appliance and a special-purpose wireless PDA made by Symbol Technologies for Sun Microsystems field personnel.

RedSonic gave demonstrations of their RED-Builder software package, an easy-to-use system image creation and deployment tool for embedded developers and showcased their real-time monitoring, diagnostic and QoS system administration technology.

RidgeRun showed off their latest version of DSPLinux, which now includes a nifty Appliance Simulator that helps developers emulate the finished product (“all the way down to the Linux framebuffer”), so they can begin developing and debugging their software long before first hardware prototypes become available.

TimeSys announced that they will deliver a complete Reference Implementation of the Real-Time Specification for Java (JSR-00001) to the Expert Group of the Java Community Process (JCP) on April 30th, 2001 for its evaluation.

Trolltech showed off their Qt/Embedded-based Qt Palmtop Environment (QPE) running on the Compaq iPAQ PDA and announced strong partnerships with three leading suppliers of embedded Linux distributions (Lineo, LynuxWorks and MontaVista).

Tuxia held a press conference to announce the US launch of their TASTE Embedded Linux operating system, which specializes in internet appliance and thin client applications. TASTE is based on Linux kernel 2.4 and a Mozilla browser, plus middleware and a full complement of plugins and other enhancements including “crash-proof” features.

Lineo's RTAI software radio earns my “geekiest ESC demo” award.

For full details on how you can replicate this experiment, see this LinuxDevices.com HOWTO article: www.linuxdevices.com/articles/AT3239582376.html.

Agenda's VR3: The “World's first Pure Linux PDA” was showcased in the Tux Theater of the ELC's Pavilion 101. It's based on an NEC VR4181.

The Intel Pro/DSL 4200 Home Gateway, shown in Red Hat's booth, is based on (surprise!) an Intel SA-110 StrongARM processor and runs Red Hat's 2.4 kernel along with RedBoot.

A yet-to-be-announced Korean cell phone/PDA

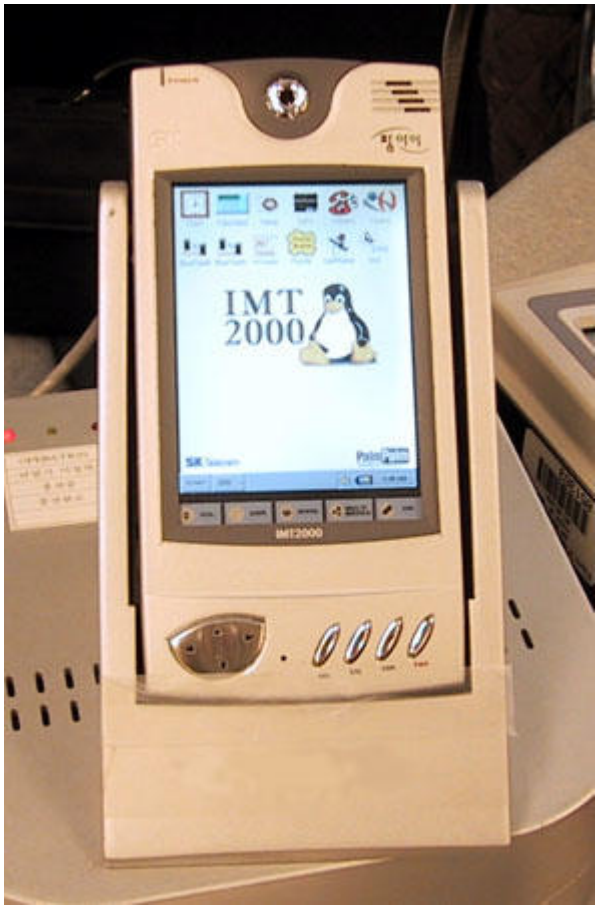
Shown by PalmPalm is the size of a *real* cell phone. PalmPalm couldn't disclose any details about the device due to being under nondisclosure with its manufacturer.

The SK Telecom PDA/cell phone

Shown y PalmPalm, is a combination PDA and cell phone that has received quite a bit of publicity over the last six months.

Ericsson's Bluetooth BLIP

Which functions as a Bluetooth access point and communications gateway, made a guest appearance at ESC in Ericsson's booth. Can you believe this tiny device contains a complete Linux system, including Bluetooth connectivity?



Ericsson's Wireless Webpad

Shown by Trolltech, has received tons of publicity over the last year. As you might guess, it uses Trolltech's Qt along with embedded Linux. The HP10 is guilt around Intel's StrongARM SA-1110 system-on-chip processor equipped with 32MB of DRAM plus 32MB of Flash memory.

Symbol Technology's wireless PDA

Shown by Red Hat, a handheld computer containing an NEC VR4181 system-on-chip processor, with 16MB RAM and 12MB Flash, and runs uClinux with Century Software's Microwindows and ViewML GUI/windowing software. Red Hat developed the required support for CDPD, GSM, 803.11B wireless connectivity. The device was developed for Sun Microsystems to be used as a handheld field service computer.

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

Archive Index Issue Table of Contents

Advanced search

Copyright Confusion

Lawrence Rosen

Issue #87, July 2001

This regular column is designed to give members of our community general legal guidance without the burden of huge attorney's fees.

The phenomenon of free and open-source software poses legal risks for software developers that challenge even experienced technology and licensing attorneys. This regular Q&A column is designed to give members of our community general legal guidance without the burden of huge attorney's fees. Send us questions about legal issues that concern you and the “geek” lawyer will try to answer them.

Do I need to put a copyright notice on my software?

—Laura Owen, Women.Com

Until 1976, copyright law imposed formal notice requirements. The failure of a copyright owner, through carelessness or inadvertence, to comply with the notice requirements caused the work to fall into the public domain. In 1976 Congress amended the Copyright Act to liberalize the notice requirement. Placement of a notice was still required, but if the owner neglected to do so, the defect could be cured by registration of the copyright and other actions within five years after publication. Then, in 1989, a new Copyright Act came into effect, bringing US law into compliance with the Berne Convention. For the first time, notice on published copies was no longer a condition of copyright protection. A copyright notice is still important, however. The Copyright Act, in section 401(d), provides that the presence of a copyright notice prevents a copyright infringer from claiming her infringement was innocent. That is, it prevents her from making the “I didn't know I was doing something wrong” defense. This has a major impact on potential damages for infringement. An “innocent infringer” may convince a court to reduce the award of statutory damages to $200, but a “willful infringer” can be held liable for statutory damages of up to $100,000. A copyright notice is easy to write. For most software, it consists of the following

three elements: the symbol ©, the word “Copyright” or the abbreviation “Copr”; the year of first publication of the work; and the name of the owner of the copyright. Make sure your copyright notice is affixed to copies in such manner and location as to give reasonable notice of your copyright. Put a notice on disks or CDs containing your software, in documentation accompanying your software and on web sites from which your software is downloaded. A proper copyright notice, even though not mandatory, can have a big payoff if you ever have to enforce your copyright.

Can a software license restrict my ability to use software?

—Harry Adams, Oracle Corp.

Most software licenses are based on copyright law. The owner of a copyright in computer software has the exclusive rights to do and to authorize, among other things, the reproduction of the copyrighted work in copies. No third party can make a copy of software without obtaining the permission of the copyright owner. The word “copy” has a specific definition in the Copyright Act: copies are “material objects, other than phonorecords, in which a work is fixed by any method now known or later developed, and from which the work can be perceived, reproduced, or otherwise communicated, either directly or with the aid of a machine or device”. That definition creates a unique problem for software. The simple loading of a computer program into memory has been held to involve the creation of a copy, one of the exclusive rights of the copyright owner. So the mere act of running a program, and thus making a copy of the software in memory, can only be done with permission - unless we can point to another section of the copyright law that expressly authorizes the making of copies to use software. Congress resolved that problem in the Copyright Act of 1976 to allow software to be used when it added, in section 117, a limitation on the exclusive rights of copyright owners. Notwithstanding the other provisions of the law, it is not an infringement for the owner of a copy of a computer program to make or authorize the making of another copy of that program provided that such a new copy “is created as an essential step in the utilization of the computer program or in conjunction with a machine and that it is used in no other manner”. To the extent that software you license is protected by copyright, section 117 of the Copyright Act provides that you are free to copy the software into memory for execution, and thus to use it in the normal fashion for which it is designed. Be careful though. Proprietary software licenses can contain other restrictions on use. Those restrictions do not stem from the copyright law, but they are imposed in a contract (e.g., the license agreement) between you and the software provider. Free and open-source software licenses do not impose such abhorrent contractual limitations on use. When you use software made available under licenses approved by the Free

Software Foundation (FSF) or Open Source Initiative (OSI), you can be confident that you can use the software freely, just as the Copyright Act provides.

Legal advice must be provided in the course of an attorney-client relationship specifically with reference to all the facts of a particular situation. Even though the answers given were provided by an attorney, they must not be relied upon as a substitute for obtaining specific legal advice from a licensed attorney.

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

Archive Index Issue Table of Contents

Advanced search

KDevelop 1.4

Petr Sorfa

Issue #87, July 2001

KDevelop provides a tool that combines the resources of contributors and existing open-source products.

- Manufacturer: Open Source Contributors
- URL: http://www.kdevelop.org/
- Price: Free Download
- Author: Petr Sorfa

The one development tool that has been lacking in the Open Source community is a professional-level IDE (integrated development environment). KDevelop thankfully provides such a tool that combines the resources of contributors and existing open-source products. However, does KDevelop match the expectations of a commercial IDE usually based on a non-UNIX platform?

What Is an IDE?

An IDE is an environment, preferably graphical, that is used for the creation, debugging and maintenance of programs. The three core components of this environment are a programmer's editor that is context-sensitive to the programming language, a GUI (graphical user interface) builder that is used to construct the graphical front end of the application and a debugger to detect bugs in the code.

These are the basic requirements of an IDE. However, there really needs to be more than these three components to make an IDE a useful tool.

Installation

Because open-source programs tend to concentrate on completing the task, rather than being user friendly, installation sometimes tends to be difficult and

frustrating, particularly considering all the different versions of Linux and the constantly changing libraries and tools.

The KDevelop RPM binary can be downloaded by either following the links off KDevelop's web site or by using a site such as http://www.rpmfind.net/ to locate it.

For this review, I installed a brand new Linux installation and made sure it included every single package and feature that the distribution allowed.

Alas, I ran into installation problems when I found certain dependencies for various libraries that did not exist in my Linux installation. A quick diversion to the Internet to download the missing libraries solved the problem.

Total installation time took about 30 minutes with a fast internet connection and a little bit of technical knowledge. This installation method is ideal for users with some Linux administration skills.

Sometimes, building from source is recommended for programmers that have non-Linux/UNIX operating systems, for customized Linux distributions and for potential KDevelop contributors. Only experienced or very determined developers should attempt building KDevelop from source code.

All the development versions of the required libraries must be installed. Because there is no easy way of determining these dependencies, building from source tends to be a process of trial and error.

A feature of KDevelop is its ability to use many existing open-source tools. Not all of these tools are required, but they are necessary to ensure that KDevelop performs as expected. When KDevelop is started for the very first time, a list of associated tools are given and are marked as either present or missing (see Figure 1). Once this list is available, the missing tools can be installed later.

Required tools utilized by KDevelop are g++2.7.2, g++2.8.1 or egcs 1.1 (I recommend g++2.9.2); make; perl 5.004; autoconf 2.12; automake 1.2; flex 2.5.4; gettext; Qt 2.2.X (which includes Qt designer and uic); and KDE 2.X.

Optional tools include enscript, Ghostview or KGhostview, Glimpse 4.0, htdig, sgmltools 1.0, KDE-SDK (KDE software development kit), KTranslator, KDbg, KIconedit and Qt Linguist. Although optional, it is best that all of these tools are available.

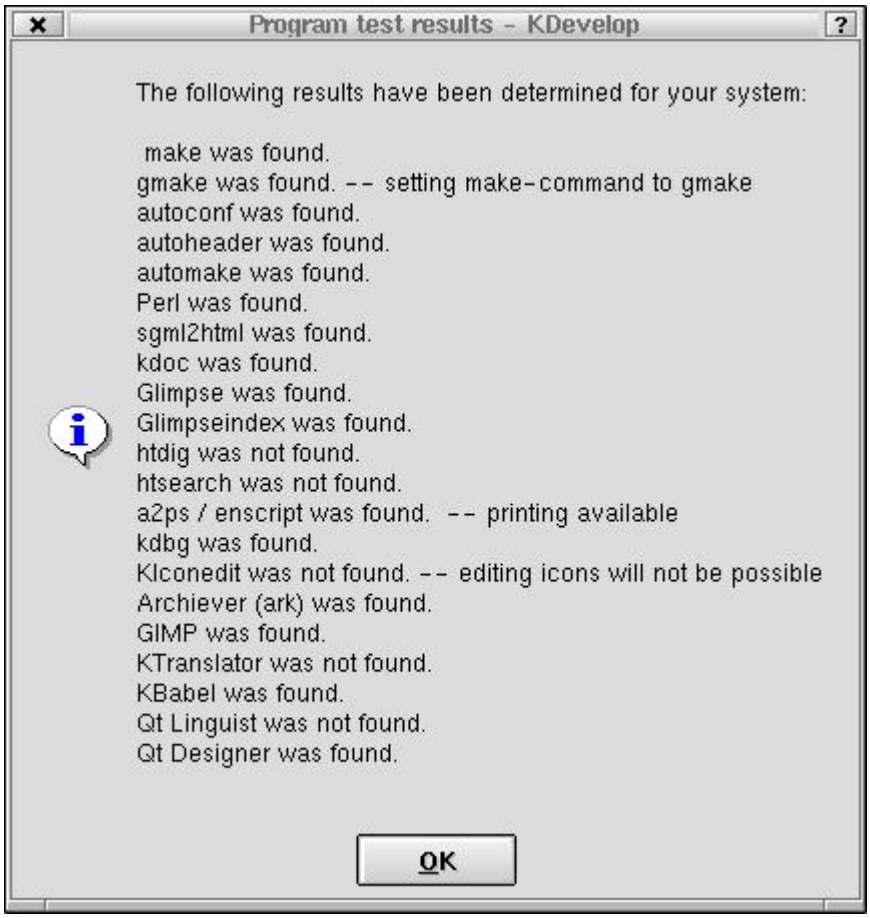


Figure 1. Initial KDevelop Startup Detecting Installed Tools

Features

Although KDevelop provides the three core requirements of an IDE (editor, GUI builder and debugger—see Figure 2), it has several other features that make it a robust and reliable tool, suitable even for commercial projects.

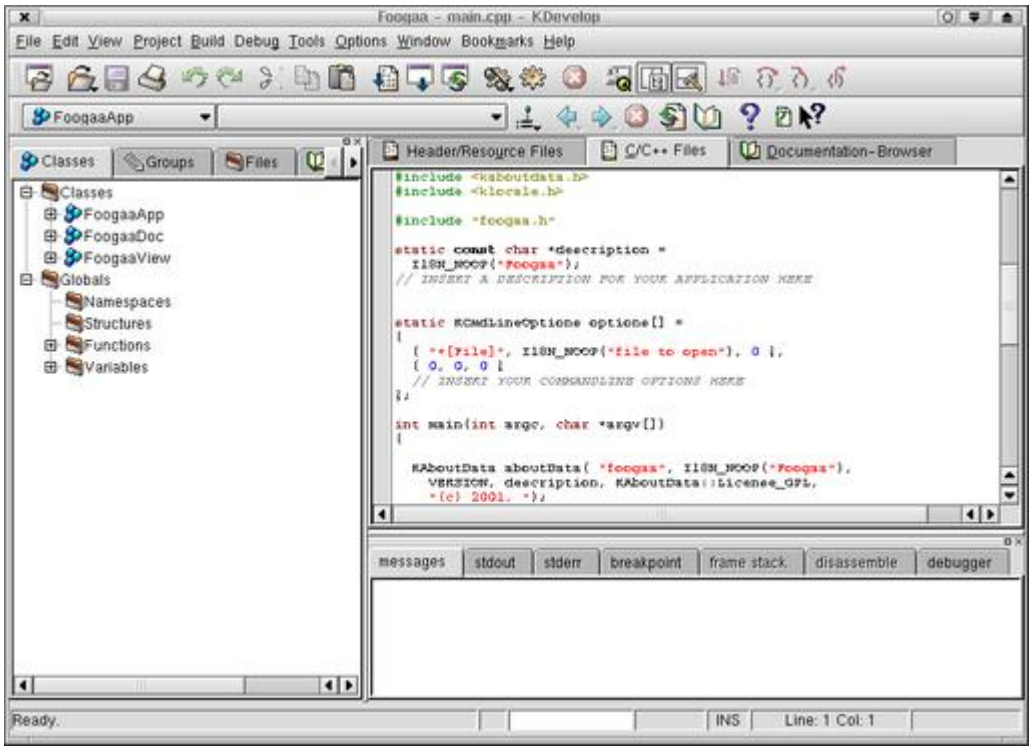


Figure 2. KDevelop 1.4 in Action

A complex program can be daunting for both beginners and experts alike; so program documentation is critical. The documentation for KDevelop provides a good source of on-line help, although it does lack screenshots and visual content. Context-sensitive help is available through tool tips and the “What's this?” cursor mode.

KDevelop also indexes the KDE Lib and Qt documentation. The ability to set bookmarks is present, which makes it easy to return to relevant documentation. Other tutorials and documentation are also available at the KDevelop's web site.

KDevelop has a built-in HTML browser that makes documentation access effortless and removes the need for an external browser.

Here are the basic interface components: Tree View, which consists of a class, groups, file, books and watch views; Output View, which provides output for messages, stdout, stderr, debugger breakpoints, debugger frame stack, debugger disassembly and debugger messages; Editor and Documentation, which includes Header/Resources editor, C/C++ files editor and documentation browser; and Tool Bar, an iconic representation of the main menu options.

KDevelop's project creation process is one of the easiest to execute using the Application Wizard, which goes through the following steps:

1. Application Type (see Figure 3)--this step allows the user to select a template for creating a program using KDE 2 Mini; KDE 2 Normal; KDE 2 MDI GNOME (Normal); Qt (Normal, Qt 2.2 SDI, Qt 2.2 MDI, QextMDI); Terminal, i.e., text (C, C++); and others (custom).

Figure 3. Application Wizard

1. Generate Settings (see Figure 4)--this is the step to enter the project name, location, initial version number, author's name and e-mail. There are also options to generate various project-associated files, such as sources, headers, GNU standard files, icons and project-associated documentation.

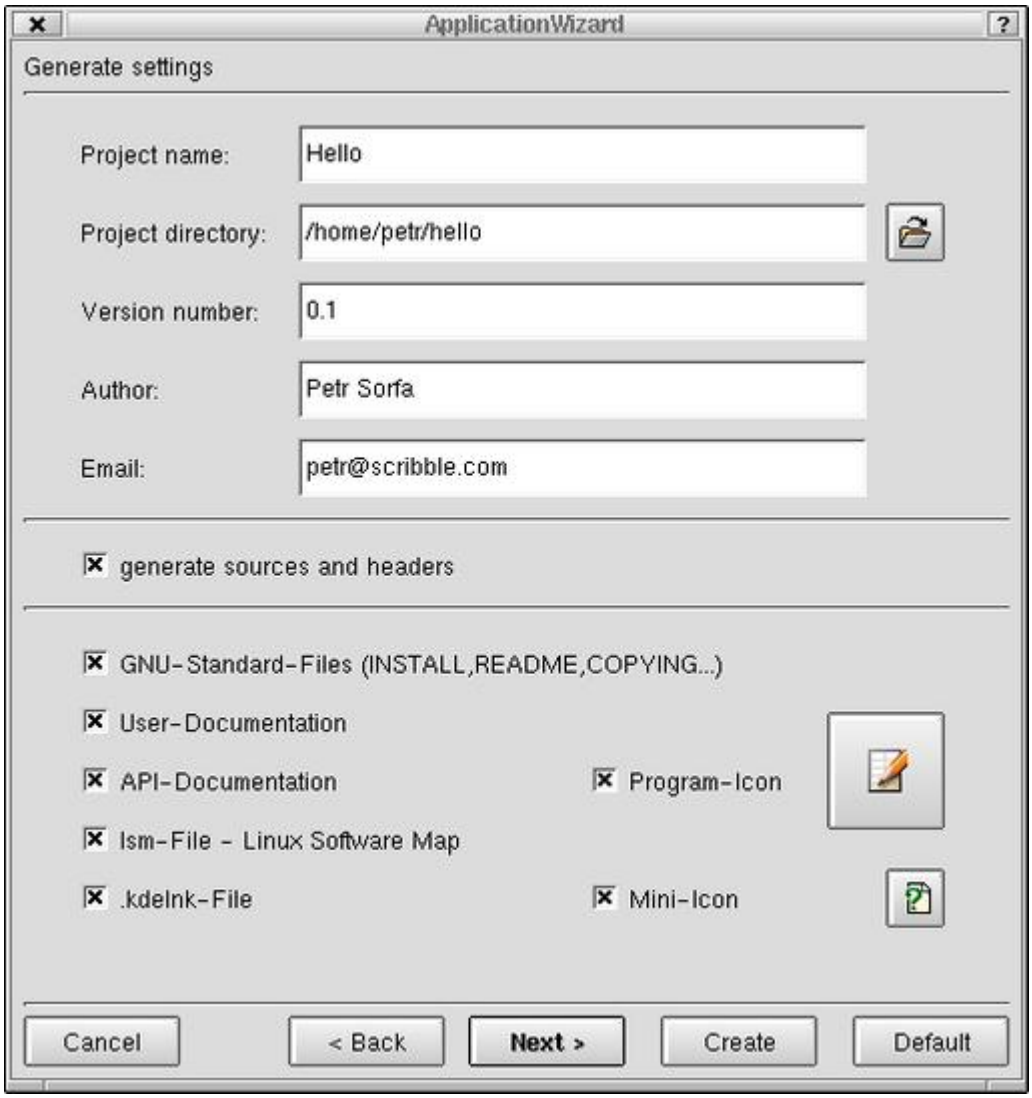


Figure 4. Entering the Project Settings

1. Version Control System (see Figure 5)--the version control system dialog allows you to set the parameters of the source control system. This is dependent on the Linux distribution. In general, this is the CVS tool.

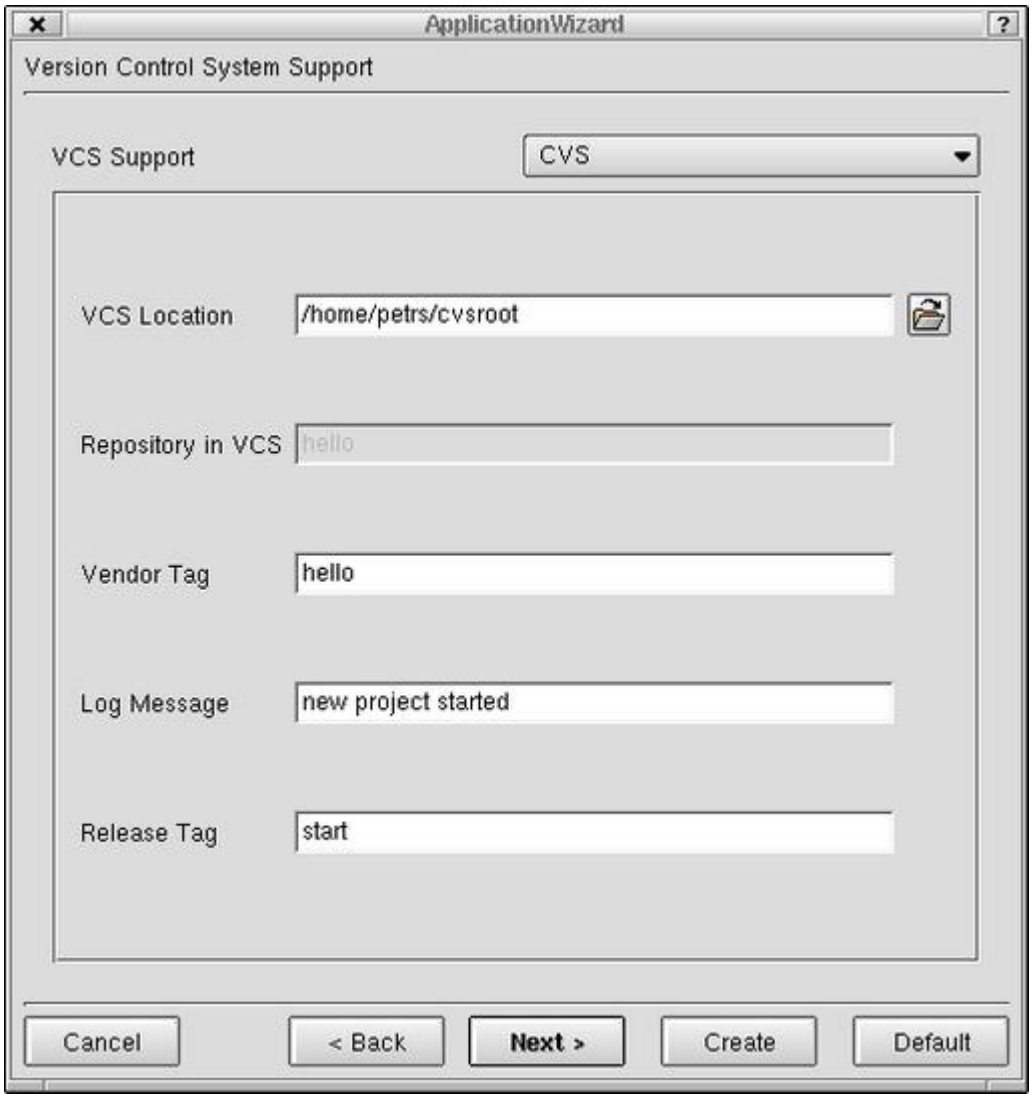


Figure 5. Selecting the Version Control System

1. Header Templates for header and code files (see Figure 6)--this allows the developer to select automatically generated headers for program headers and source files. These headers are fully customizable with tag expansions, which fill in various bits of information, such as the author, filename and date.

Figure 6. Header Template Setup for Header Files

1. Project Creation (see Figure 7)--in the final stage of project creation, the related project's files and directories are created, using the **automake** and **configure** tools. Note that if some of the required tools are missing in the Linux distribution, this creation process might fail. If failure does occur, it is best to install the missing components and then recreate the project. It is extremely difficult to recover from a project-creation failure.

Figure 7. Initial Project Creation

Once the project has been created, development can begin. I strongly suggest that at this point the project be built and executed to detect any build problems.

Qt Designer

KDevelop 1.4 uses Trolltech's Qt Designer. Qt Designer provides a professional interface, allows GUI building with most of the Qt widgets and is a very useful tool for relating GUI widgets and components with each other (best thought of as visual programming).

Here is a synopsis of the process to create GUI components with Qt Designer under KDevelop 1.4 (see Figure 8).

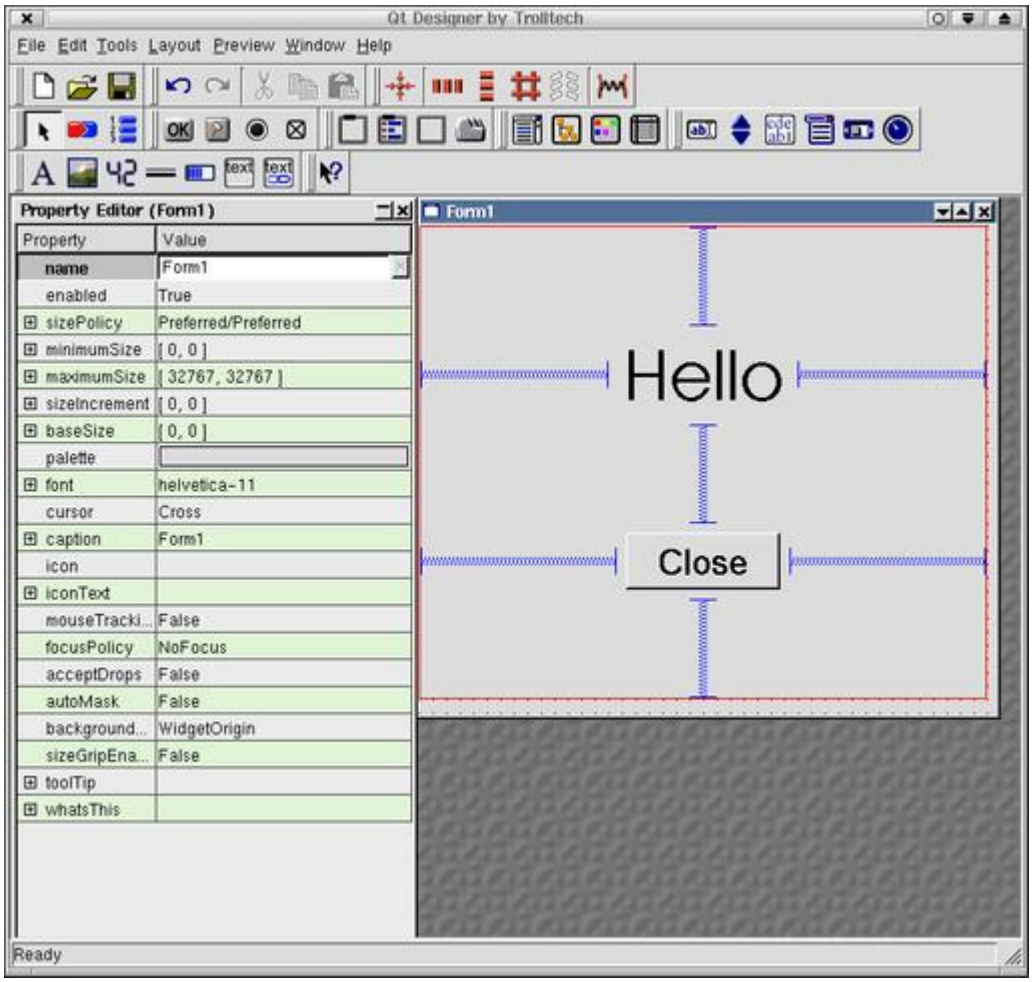


Figure 8. The KDevelop 1.4 GUI Builder, Trolltech's Qt Designer

Qt Designer is a major product itself and requires a separate article to fully describe its capabilities and usage. In this article, its relevance to KDevelop will be covered.

Qt Designer allows the use of layout tools and access to all widget properties. It has the ability to create relationships between widgets, such as the click on a push button with the closing of a window.

Qt Designer only generates an intermediate XML .ui file describing the dialog. Another Qt utility, **uic**, is used to generate the actual source code files from the .ui file. KDevelop 1.4 supports the .ui files, but the user needs to add the .ui file to the project. When the user initiates a make or rebuild, KDevelop automatically calls uic to generate the relevant associated code.

Unfortunately, uic rewrites all the generated code files whenever the user changes the .ui file with Qt Designer. This implies that the user cannot edit these generated source files. To use the code generated by the uic tool the user needs to inherit the generated code classes before implementing the user-defined functionality.

Debugging

KDevelop harnesses **gdb** in order to provide debugging facilities (see Figure 9). Clicking in the left-hand column of the editing windows sets a breakpoint in existing code. The breakpoints can even be set when the program is not running or is in a noncompilable state, which are known as lazy breakpoints. KDevelop displays lazy breakpoints in blue and active breakpoints in red. A little green arrow next to the corresponding line of source indicates the current point of execution.

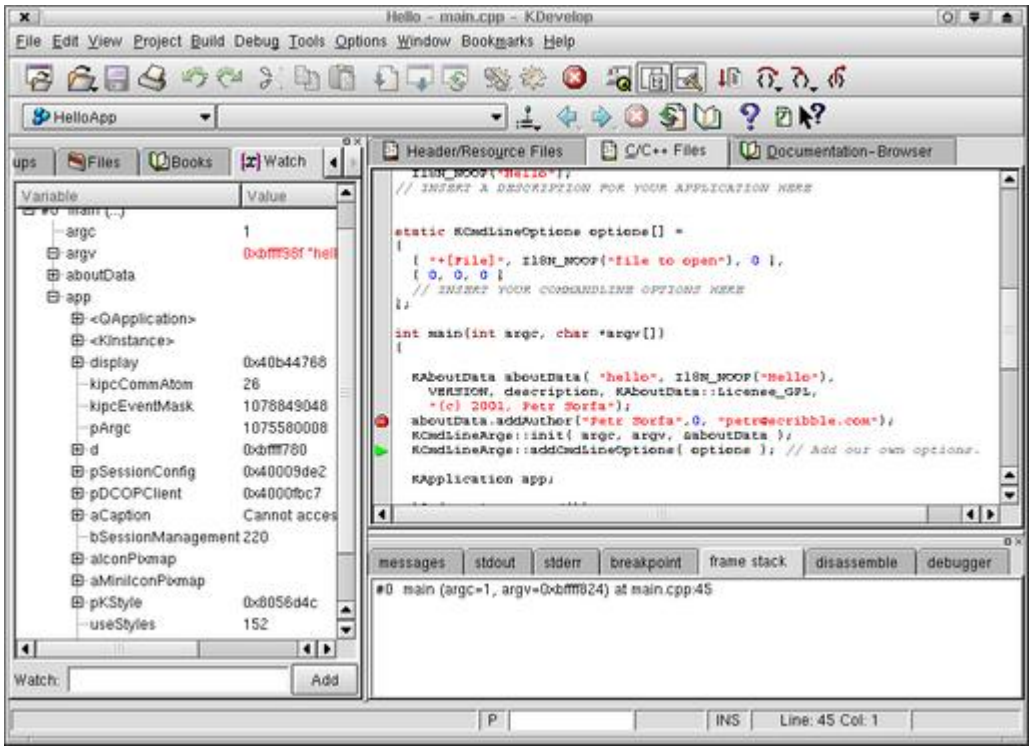


Figure 9. Debugging the Project

KDevelop provides most of the required basic debugging functionality, such as basic execution, next line and program interruption. The user can activate a floating debugging toolbar for easier debugging command access (see Figure 10). In the tree views, the variable tree tab displays the currently available variables.

Figure 10. The Floating Debugger Toolbar

Debugger-related information is displayed in the debugger, assembly, frame stack and breakpoint output windows.

The problem with the default debugger support is that users wishing to do finer-level debugging cannot access gdb directly. Another problem is that the user can alter variable values only in a non-intuitive way via the Watch input line. However, KDevelop can be configured to use an external debugger, such as the ever-popular **DDD**, **kdbg** and **xxgdb**.

External Applications

KDevelop allows the executions of KDE applications within its framework. Applications such as the **GIMP**, **Ark** and **KBabel** are set up by default. Users can add their own via the options->tools menu.

Compiling, Building and Distribution

Compilation and building of the project can be done through various menu options, such as make, clean, rebuild, clean for distribution and auto configuration. KDevelop is intelligent and will prompt you, if necessary, to rebuild the project before program execution.

The Project->Make Distribution->Source.tgz menu item allows the creation of the source for distribution. Unfortunately, there does not seem to be a way of automatically generating RPMs or RPM spec files for more useful packaging.

Configuration Management (Source Control)

If version control system was chosen during the project's creation, designated files can be added to source control system (see Figure 11). This can be done by selecting the file's Add to Repository pop-up menu option in the Group or File view. Changes can be committed via the Commit option and other developer changes retrieved with the Update option.

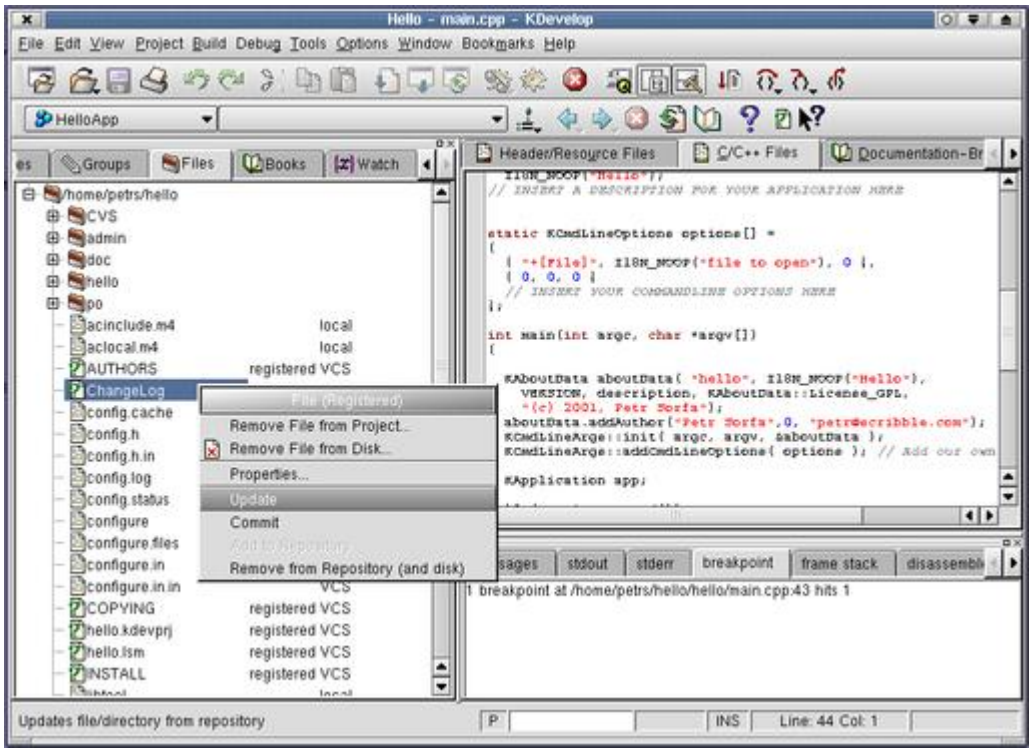


Figure 11. Using the Source Control System

Because CVS supports remote repositories, it is possible to have multiple developer projects using KDevelop. However, KDevelop does not provide the full functionality associated with CVS, such as file watching and editing privileges.

User Generated Documentation

KDevelop has hooks for generating program API documentation via **kdoc** and **doxygen**. When generated, the user can browse the user API documentation with KDevelop. This is very handy for large projects with several developers.

If user documentation was selected during the creation of the project, a user manual HTML template is automatically generated. It is up to the user whether to use an HTML editor to fill out this information.

Support

One of the possible disadvantages of open-source projects is support. Occasionally a project goes into hiatus, and it might be virtually impossible to contact someone concerning problems, help or bugs. However, KDevelop has a very active mailing list, which is continually monitored by the several maintainers of KDevelop. KDevelop itself provides a bug-reporting tool that allows users to send problem descriptions to the KDevelop folks.

Therefore, support is not a problem, and coupled with a good range of on-line documents, KDevelop provides a level of support that most commercial products cannot match.

What Is Missing Wish List

Although KDevelop is a robust and useful tool, several functional areas are missing or still need to be improved:

- A smart editor would be handy that would automatically complete your code, like the parameters for the current function.
- KDevelop 1.4 language support is limited mainly to C++ and C applications using the gcc/g++ compiler.
- There could be better support for integrating with other GUI builders, such as the GNOME GUI builder, **glade**.
- Incorporating an existing project into KDevelop is not easy.
- Rapid application development (RAD) components that provide database connectivity and a base for enterprise level development are not present.

Because KDevelop is an open-source program, these missing or incomplete features may not be such problems after all. The KDevelop team is continually striving to improve the IDE, and if a feature is really wanted, implement it yourself and be part of the KDevelop team.

Summary

KDevelop has the capabilities equivalent to an intermediate level commercial IDE. It integrates well with the Linux platform, makes use of many open-source tools and provides a level of support that is hard to beat. Although there is still room for improvement, KDevelop fulfills the functions of a development environment suitable for small to intermediate projects and development teams.

Petr Sorfa (petrs@sco.com) is a member of the Santa Cruz Operation's Development Systems Group where he is the maintainer of the cscope and Sar3D open-source projects. He has a BSC from the University of Cape Town and a BSC Honours from Rhodes University. His interests include open-source projects, computer graphics, development systems and sequential art (comics).

Archive Index Issue Table of Contents

Advanced search

Catching up with KDE

Robert Flemming

Issue #87, July 2001

Currently supporting 34 languages, KDE is poised to answer many of the questions surrounding Linux' viability on the desktop.

- Manufacturer: The KDE Project
- E-mail: kde-user-request@lists.netcentral.net (users' mailing list subscription)
- URL: http://www.kde.org/
- Price: Free Download
- Reviewer: Robert Flemming

KDE developers may be one step closer to “konquering” the desktop with the most recent 2.1.1 release of the K Desktop Environment (http://www.kde.org/). The development cycle has intensified since the 1.0 series, bringing new features and stability improvements to users at an ever-increasing rate. In fact, as of this writing, the first alpha version of KDE 2.2 has been released for testing. End users and developers alike will benefit from the newest offering. Currently supporting 34 languages, KDE is poised to answer many of the questions surrounding Linux' viability on the desktop.

In addition to stability enhancements, the latest release includes a large number of cosmetic improvements that create a more unified and polished interface. Kicker, the 2.0 replacement of KPanel, received a number of new features as well as the return of an old one. For all of those WindowMaker users who just can't bare to part with their beloved dock applets, yearn no more. Kicker is now able to swallow your favorite applets into a new dock application bar (see Figure 1). Support has also been added for child panels, and the external taskbar noticeably absent from version 2.0 has returned, along with theme manager. Despite the re-inclusion of the theme manager, a lack of integration between the various theme-able elements of the desktop is still present. Widget styles, icons, colors, backgrounds and KWin decorations

each need to be managed from their own individual Control Center modules. The work of the KDE artists team is not be overlooked, however. A number of icons were added and improved, and the new splash screens serve to unify desktop applications (see Figure 2).

Figure 1. The New Dock Application Bar

Figure 2. Desktop Splash Screens

Perhaps one of the most significant advances in the latest version is anti-aliased font support. In order to enable anti-aliased fonts, KDE must be built upon QT 2.3, in combination with XFree86's Xft extension. Linux has been somewhat plagued by issues regarding true-type fonts and anti-aliasing; while these advances greatly improve that, setting up anti-aliased fonts can be a bit of a challenge. Instructions can be found at trolls.troll.no/~lars/fonts/qt-fonts-HOWTO.html. If you installed KDE via packages, it may not have been built using the necessary libraries. If this is the case, you should check for newer package versions prior to trying to configure anti-aliased fonts. Figure 3 shows an enlarged comparison view of a web page as viewed from within Konquerer with anti-aliased font support and Netscape without anti-aliased font support.

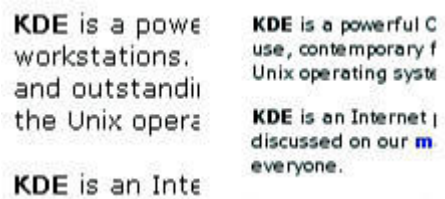


Figure 3. Fonts with and without Anti-Alias Font Support

Without a doubt, the gem of the whole project is Konquerer, KDE's next-generation web browser, file manager and document viewer. Konquerer's modular architecture allows for easy extendibility to support current and emerging internet technologies, such as HTML 4.0, Java, JavaScript, XML, Cascading Style Sheets (CSS-1 and CSS-2) and SSL. In addition to built-in components, Konquerer is able to utilize existing Netscape plugins to provide support for Flash, RealAudio/Video and other multimedia programs. Taking a page from Eazel's Nautilus, Konquerer added support for text preview, a feature that creates a thumbnail of the first few lines of a text document within the icon. When used in combination with the experimental alpha blending support you are able to get a semitransparent representation of the MIME-type icon under the text. In addition to text and image preview, Konquerer is also capable of generating thumbnails of HTML documents (see Figure 4). HTML files are read and thumbnailed in the background after loading the directory view, allowing for a smooth browsing experience.

Figure 4. HTML-file Thumbnails

The mechanisms by which Konquerer becomes more of an application framework than a file manager or web browser are called IO Slaves. IO Slaves are small bits of code based on the KIO library that know how to send and receive data using a specific protocol. From its inception KDE has focused on network transparency, and the KIO architecture has allowed for the creation of a variety of special purpose plugins that enhance the capabilities of Konquerer. Seamless support exists for browsing the Linux filesystem, NFS shares, MS Windows shares, web pages, FTP sites and LDAP directories, to name only a few. With a minimal amount of code, developers can make available new protocols and conduits to all KDE applications. Some of the other IO Slaves developed or being developed include digital camera support via Gphoto2, which allows for drag-and-drop access to images stored on your digital camera, and enhancements to the existing audiocd plugin that allow for drag-and-drop CD ripping and MP3/Ogg Vorbis encoding. Many of Konquerer's IO Slaves exist without much fanfair, yet are quite useful. For example, entering **man:/df"** or **"#df** into Konquerer will produce the man page for **df** (or other command of

your choosing). For those preferring the GNU Info documentation browser, give **info:/df** a try. For quick access to your floppy drive, enter **floppy:/** into the location bar. Another little-known addition to Konquerer is the new shell command function. For example, while browsing a directory via Konquerer, pressing Ctrl-e and entering a shell command like **du** will present the user with a dialog box displaying the output of the command in the currently viewed directory. A list of currently installed IO Slaves is available via the information section of the control center.

One particular IO Slave recently introduced pertains to LAN browsing. Those coming from a Windows background will most likely compare this to the Network Neighborhood. At the heart of this new feature is LISa, or the LAN information server. Unlike the Windows' Network Neighborhood, LISa only relies upon the TCP/IP stack and no other protocols such as SMB or NetBIOS. In short, once you configure LISa with information about the network to which you are attached, it will probe devices found on the network for commonly available services, such as FTP, SMB, NFS and HTTP (see Figure 5). By entering **lan:/** into Konquerer, you will be presented with a list of all discovered servers and their associated services. While it may sound like LISa would create a lot of unnecessary network traffic, that is not the case. In fact, the more clients you have on your network running LISa, the more efficient it should become. LISa itself is a dæmon that runs on the client. Upon startup, that dæmon sends out a broadcast in an attempt to discover already existing LISa servers on the network. In the event that one is found, the network servers list is transferred to the new client without unnecessary network probing. At any given point in time, there should be only a single LISa node on the network doing the actual probing. LISa needs to be run as root and can be configured via the Network/ LAN browsing control center module. LISa is a component of the kdenetwork package.

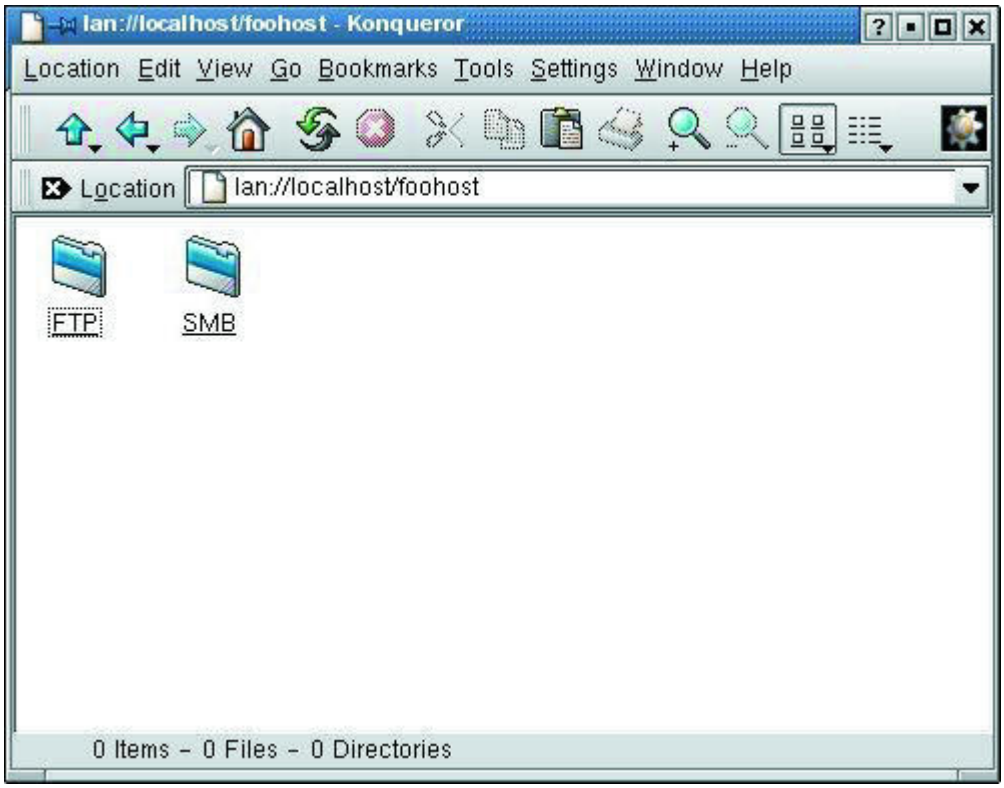


Figure 5. LISa Searching for Services

For the developer, KDE provides a rich set of tools for application development. Among these technologies are the desktop communication protocol (DCOP), a component object model (KParts), an XML-based GUI class and the previously mentioned I/O libraries (KIO). Tying all of these various [development] tools back into the desktop is the coordinated release of KDE's advanced IDE/RAD, KDevelop 1.4. Multimedia components are handled via an architecture built upon the network-transparent analog real-time synthesizer (**aRts**).

DCOP is the much talked about client-to-client communication protocol that replaced CORBA early on in the 2.0 development cycle. DCOP is built upon the standard X11 ICE library and presents a faster and more lightweight interface than what was previously being developed with CORBA. KParts, KDE's component object model, is what allows applications to share components and embed themselves within one another. The most extensive use of this technology can be seen in KOffice and Konquerer. Utilizing XML as a method of creating GUI elements dynamically, developers are able to provide a more customizable and standardized desktop interface. To aid desktop constancy, KDE has worked at establishing coding standards and a GUI styleguide. Since GUI elements are generated dynamically, updates to the styleguide are reflected immediately across GUI elements without recompilation or modification. **aRts** utilizes a CORBA-like network design enabling remote applications to output sound to the local workstation, providing a multimedia compliment to the network-transparent features provided by XFree86 and KIO. For more information on KDevelop or KDE application development, see the

KDevelop 1.4 review on page xx. A wealth of developer-related information, including tutorials, FAQs and standards guides, can be found at http://developer.kde.org/.

It's astonishing to think of the strides Linux has made as a desktop operating system over the past few years, and the latest offering from the KDE camp is indeed a testament to this progress. With the emergence of new companies, as well as older, more established ones, focusing on improving and providing applications for the Linux desktop, greater acceptance is likely not far behind. The Kompany (http://www.thekompany.com/) has been turning out an amazing number of much-needed Linux applications. IBM has been working with Trolltech on integration of their ViaVoice software into QT to provide speech recognition to Linux users. KDE development as a whole is moving at a rate faster than ever before. Each release brings Linux one step closer to coming out of the data-center and onto the desktop.

The Good/The Bad

Robert Flemming is a network administrator at VA Linux Systems, and you'll have to pry Konquerer out of his cold, dead hands. Questions and comments may be sent to flemming@valinux.com.

Archive Index Issue Table of Contents

Advanced search

Letters

Various

Issue #87, July 2001

Readers sound off.

MPEG Unpegged

I've been deeply involved with MPEG for several months now, so your article on MPEG-1 playback programs caught my eye (*LJ* May 2001). I found several factual errors about MPEG in the article.

First, MPEG-2 video does not necessarily have better video quality than MPEG-1: the differences between the video portions of the MPEG-1 and MPEG-2 standards are fairly minor. An MPEG-1 file with a 720 x 480 frame size, compressed to 6 Mbit/s is legal, and will be very similar in appearance to a 720 x 480 6 Mbit/s MPEG-2 movie.

Second, the author states that “not all MPEG-1 files are entirely `compliant”'. The MPEG standards define what a compliant decoder is expected to be able to handle, but not how a compliant encoder works. In other words, a compliant decoder is not one that is “tolerant”, but is instead one that adheres closely to the letter of the ISO 13818 standard, so as not to be surprised by the output of a novel (but legal!) encoder. Unfortunately for free software authors, these documents are expensive: the basic set of MPEG-2 PDFs (ISO 13818-1 through -3) costs $424, and the complete set is $1,390 (from http://webstore.ansi.org/). It's little wonder that commercial MPEG products far outstrip free ones' capabilities, in nearly all cases.

Third, the author states that an “MPEG-1 audio stream...is an MP3 file”. This is not always true, and is not even likely. MPEG-1 defines three different audio encodings (or “layers”). Layer I audio is the most basic, but it isn't used very often. Layer II is the most common: video CDs and MPEG files you download from the Internet almost always use Layer II audio. Layer III (aka MP3) is optional, so most MPEG decoders don't include support for it. Since few

decoders support Layer III audio, most encoder creators also don't bother including support for it.

—Warren Young

Certified Sword

Just a note to let you know that the certified sword cuts both directions (Editorial focus, *LJ* May 2001). As CIO of a multimillion dollar corporation it is my job to not only run the IS department but to hire and fire employees. One of the first questions I ask is if the prospective employee has any certifications, if so I politely tell them “I'll let you know if we can use you” and promptly throw their application into the garbage can.

Before you ask, no I am not certified. I have however taught pre-MCSE classes at Unisoft Institute of Technology in Houston and was horrified to learn that I had to teach the way the test worked, and the way the real world worked (pre-MCSE in this case really meant A+ and Networking certification). This effectively meant that I held two classes in one, which to say the least was difficult. Additionally, long ago before computers were my profession I was an ASE-certified mechanic. Since I have passed 10 ASE certifications I can tell you that they are just as much a joke as computer certifications. I quickly realized that even holding all the certifications I did, and after graduating from a top automotive technical school with a 4.0 GPA and Alpha Beta Kappa National Honor Society, that I was not a very good mechanic.

To me, being certified means that the person does not have enough knowledge or experience to get the job on their own merits and hopes that this piece of paper will help them, and it does not. In my experience the only time certifications help you is when you are applying to a business where the person responsible for creating hiring policies is not a real computer technician.

I am forced to deal with MIS and IS degrees from recent college graduates as well as a plethora of certifications on a regular basis. Unfortunately, I have found that the people who have neither a degree nor a certification but who have been working with computers for ten years are much better equipped to handle the job. At least if they are inexperienced I can teach them the way things really work instead of attempting to retrain them after they have their degree or certification.

—Allan Hall

Carried Away with Gratitude

I very much enjoyed the April 2001 *Linux Journal* article “Linux on Carrier Grade Web Servers”. You did a nice job of describing the software choice, hardware environment and test results. I look forward to future articles discussing the other LVS implementations (direct routing and IP tunneling) and comparing their stability and performance with that of the NAT implementation.

Thank you.

—Bill Landahl

Some Saucy Suggestions

I enjoyed your articles on Linux Certification (*LJ* May 2001). I thought the “real-life” experience was very telling, although perhaps toned-down a bit to protect the vendors.

Here's my thoughts on what I read:

We can earn an extra $10K per year by becoming certified? Really? Who can? New grads? Having worked on, supported and/or maintained SVR4, AIX, HP-UX and Solaris for twenty years, I can't imagine getting another $10K just because I had some Linux certificate.

I looked at some of the questions from Red Hat's and Sair's study guides and tests. What a crock! “What's the fdisk type code for a Linux swap partition?” Who cares?! Look it up by typing **l** to list the types. Forgot that command? Type **?** to list them all. Better yet were the impossible to understand questions and answers on Sair's test. Their “correct” answer for **wc -l *** is that it returns the total number of lines in the files. Gee, my experience is that it shows the line count for each file, followed by the total, but that answer isn't available. Yes, they want the entire Linux community to help improve the tests, but if they can't get the simple things right, I'd hate to see how they do with the hard topics.

Finally, the exam companies could learn a good lesson from the FCC and ARRL. The amateur radio exams are also multiple choice, but they are composed of a certain number of sub-elements. Each sub-element has a number or required topics. Each topic has a number of published questions and answers, with references to the rules and regulations. The effect is, the actual exam might have only 25 questions, but those questions are pulled from a pool of several hundred, and each critical element is covered.

—Mike Hall

Thanks for the GRUB

Mr. Marshall,

I want to extend a gigantic thank you for the article in the May 2001 issue of *Linux Journal* on the GNU Boot loader, “Boot with GRUB”. It could not have arrived at a better time.

We have a Linux machine whose main (boot) hard-drive started giving us IDE bus resets and other attendant errors. It became unusable although we are pretty sure the files we need are probably still good.

I tried constructing a new system from scratch and then copying the needed files and applications to the new system. Unfortunately this didn't work.

After some effort I was able to clone the bad drive onto a similarly-sized replacement drive using Norton's Ghost program. However, the LILO booter was no longer functional.

With the use of the information in your article I was able to construct a boot floppy that would get the replacement drive booted, and then I ran LILO on it to get the boot configuration properly re-written onto the drive. It is now a booting system.

—Keith Ericson

What Do You Expect?

In the “Best of Technical Support” May 2001 a suggestion was made to use the command **expect**. The editor inserted a note that expect was described in an article published in the December 2000 *Linux Journal* but did not mention the specific article, author or page number. I relied on the Interactive Journal to find it. However, expect did not get picked up by your search engine. I ended up searching each article with my web browser's find feature and did locate the article: “Linux System Administration: A User's Guide” by Marcel Gagné. May I suggest that when referencing an earlier article that you use a fuller citation. Thanks.

—George Palma

In addition to Marcel's article, we've run two other articles on expect. One can be found in issue 54, “Automating Tasks with expect” and one in 68, “What Can You Expect?”

—Editor

History Lesson

I would be more inclined to take Tobin Maginnis' infomercial on Sair Certification, “Why be Certified”, seriously, if his grasp of PC history wasn't as shaky as his understanding of Shakespeare (*LJ* May 2001).

When IBM introduced the original PC, it didn't “revolutionize the technology”. The design borrowed pretty heavily from the Apple II, and for the first few years of its life, it was a fairly pathetic machine. However, it had the one magic component, those three letters on the label. That made it socially acceptable in the office, even though it was distinctly inferior to the CP/M machines of the time. (On the positive side, it brought an end to the bewildering proliferation of floppy disk formats then current.)

Ever since Novell hit on the concept of certification as an extra cash cow, and corrupted the term “engineer” in the process, it's been making life easier for ignorant personnel (aka HR) types to sort resumes into piles, and I don't suppose that's going to go away. The only question is, to which pile will this certificate direct my resume?

—Alan Rocker

Deep Pockets

I received the latest issue with the Training & Certification focus and was very disappointed to read that Red Hat's RHCE program was given essentially no print space, despite being widely recognized as the industry leader in Linux certification.

Having just completed the course, I can say that it is an accurate measure of a person's basic Linux systems administration skills, and even with 6+ years experience administering Linux, I found the class informative and the exam challenging.

I find the lack of mention of the course very disturbing, and I can only hope that this is not the beginning of a trend that will see *LJ* catering towards advertisers with deep pockets rather than accurately reporting on the Linux world.

—Cheyenne T. Greatorex, RHCE

Cheyenne, There certainly is no such trend. Between Sair, LCI and Red Hat I would have to say that the latter has the deepest pockets.

—Editor

UpFront

Various

Issue #87, July 2001

Stop the Presses, *LJ* Index and more.

Buzz Match: Who Does What?

See if you can match each of these companies with their buzzphrase description of themselves. All buzzphrases are copied and pasted out of each company's own press releases or corporate boilerplate.

For more fun, random-generate your own buzzphrases at http://www.BuzzPhraser.com/. And if you don't like that engine, take the source code and build your own. It's free and open.

—Doc Searls (*Linux Journal*'s leading expert buzzware management solutions provider)

1) Red Hat	a) Linux-based software solutions for the Internet and enterprise computing infrastructure
2) Caldera	b) the leading CyberSecurity product, service and training solutions provider
3) Linuxcare	c) a leading provider of software and services for connected smart devices
4) VA Linux	d) the expert provider of Linux and open-source solutions for the Web
5) APC	e) the "Unifying UNIX with Linux for Business" technology leader in developing and marketing successful Linux-based business
6) Chek	f) the leader in developing, deploying and managing solutions built on the benefits of an open-source platform
7) Aberdeen	g) the world's leading supplier of business-to-business embedded computing platforms for use in telecommunications, network storage, imaging, medical equipment, and semico equipment applications
8) Mainsoft	h) a leading developer of scalable messaging and e-mail infrastructure software for Internet Service Providers, Application Service Providers and corporations
9) Bay Mountain	i) a leading provider of state-of-the-art application development technology and business solutions
10) IBM	j) a leader in providing comprehensive professional services and solutions for Linux and open-source technologies
11) Trustix	k) a leading application infrastructure provider
12) Zero G	l) a leading provider of web and application infrastructure services
13) SecureInfo Corp.	m) leading provider of global, end-to-end

```
                              availability enhancement solutions
  14) Motorola Computer    n) the leading provider of
      Group                   open-source e-commerce applications
  15) Wind River           o) a leading provider of communications
                              infrastructure software for service providers
  16) Rockliffe            p) a leading market analysis and positioning services
                              firm
  17) Magic Software       q) the e-porting company
  18) TurboLinux           r) the leading independent software vendor of
                              network management solutions for Linux
  19) Zelerate             s) the leading global provider of IT solutions
```

Answers: 1-f, 2-e, 3-j, 4-d, 5-m, 6-o, 7-p, 8-q, 9-l, 10-s, 11-r, 12-k, 13-b, 14-g, 15-c, 16-h, 17-i, 18-a, 19-n.

NASA's JPL Builds War Game Simulator on Linux

The Jet Propulsion Laboratory (JPL) of Pasadena, California is one of the space program's major players. Managed for NASA by the California Institute of Technology, JPL is the lead US center for robotic exploration of the solar system, and its spacecrafts have visited all known planets except Pluto. In addition to its work for NASA, JPL conducts research and development projects for a variety of federal agencies. One such project, the Corps Battle Simulation (CBS), recently made the transition from VAX to Red Hat Linux 7.0, resulting in a substantial increase in performance at considerably reduced cost.

CBS has been used to train army officers in battle tactics for over 15 years. Previously, it ran on VAX's most powerful computer, a $100,000-plus 7800-series machine. However, due to the steadily increasing intelligence and the addition of new features, CBS reached its limitations on VAX. This made further innovation a struggle and threatened to render the battle simulator obsolete within a few years. As a result, the US Army's Simulation, Training and Instrumentation Command (STRICOM), in Orlando, Florida asked JPL to port the software to Linux in order to increase functionality while cutting cost.

After spending a man-year reconfiguring CBS source code, then recompiling, testing and debugging, the team benchmarked the system running on Linux with rewarding results. “By porting CBS from VAX to Linux, we have achieved far better performance at a much reduced cost and have lots of extra capacity”, says Jay Braun, a simulation software technologist at JPL.

The additional capacity of Linux gives the CBS system more room to expand. Terrain elevation, for instance, can now be modeled to a highly detailed level. Previously, attempting complex line-of-sight calculations severely taxed VAX capabilities. Now, high-fidelity maps are available on Linux that make simulations more realistic, increasing the accuracy of the battle scenarios.

CBS is running on a $4,000 PC with a 1.2GHz AMD Athlon processor. This Linux machine runs the largest CBS exercise almost four times faster than the most

powerful VAX, without sacrificing anything in model fidelity. Using the VAX, fidelity had to be reduced in order to allow a simulation to progress at a one-to-one game ratio, i.e., a virtual minute in the simulation requires a real minute of execution time. Under Linux, however, one-to-one scenarios can be achieved at the highest quality levels available.

JPL has also made adjustments so that CBS has a 20-second save time for the largest exercises and three seconds for small exercises. This is an order of magnitude faster than the old VAX system. Under Linux the application can now represent almost 3GB of virtual address space for each simulation. “That's a big image!” says Braun. “Our model has plenty of features that are pushing the limits of Linux.”

JPL will deliver the ported software in June 2001. Braun predicts that in the near future, the system will further advance to a two-processor machine that can support additional simulations. JPL is now shifting over to Red Hat Linux 7.1 with the new 2.4 kernel.

—Drew Robb

Now Everybody Knows Where You Live

Wonder what the weenies at Google are up to, besides finding ways to make 17,000+ Linux servers search for everything in nothing flat? Try finding out where you live. It's easy. Maybe too easy.

Substitute your name for these: John Doe KY (in other words, first name last name two-letter-state-abbreviation). If they get enough information from some white pages directory, they might even come up with a Yahoo! map to your house.

Want to de-list? Go here: www.google.com/help/pbremoval.html.

And, for more information, go here: www.google.com/help/features.html#wp.

—Doc Searls

LJ Index—June 2001

1. Uptime in percentage claimed by Chek: 99.928
2. Uptime in percentage claimed by some Microsoft ads: 99.999
3. Billions of unique lines of C/C++ software code eligible for migration to the Itanium 64-bit platform: 100

4. Percentage of Kuro5hin readers who watch TV less than one hour a day or not at all: 65
5. Sum SuSE is charging high school students for its Linux distribution: 0
6. Number of Linux boxes SuSE is initially sponsoring for high schools in the US: 2,000
7. Billions of dollars professional venture funds invested into new startups over the past two years: 160
8. Millions of hits per day at the Apache.org web site: 2
9. Peak bandwidth demand on the Apache.org web site in Mb/sec: 15
10. Number of sites in millions found by Netcraft to be serving with Apache: 17.238
11. Number of Jabber servers: 35,000
12. Millions of wireless shoppers by 2004: 373
13. Number of Americans in 70 now on the Wireless Web: 1
14. Number of Americans in 3 expected on the Wireless Web by 2005: 1
15. Percentage of Cingular's 20 million cell phone customers that access the Web: 50
16. Range in billions of dollars on wireless ads by 2005: .89-6.1

Sources:

- 1: e-mail from Chek
- 2: Microsoft advertising
- 3: Aberdeen Group, www.migratec.com
- 4: Kuro5hin.org
- 5-6: SuSE
- 7: Red Herring
- 8: Brian Behlendorf, speaking to the Apache Software Foundation Meeting in April
- 9-10: Netcraft www.netcraft.com
- 11: Jabber.org
- 12-16: Graeme Thickens, reporting on what was said at an Industry Standard conference on Wireless. David Weinberger adds, “Attending the conference were between 200 and 300,000,000 people.”

Apache Keeps Rocking

Maybe it was the long-awaited Apache 2 beta release in March, or maybe it was the “increasing returns” economics by which the huge get ubiquitous while the small get trivial. Any way you look at it, it's hard to beat the increasing majority

Apache—which is open source—enjoys as a server of web content to the World.

Netcraft's April 2001 survey finds nearly 18 million sites serving with Apache, or 62.55% of the total population of 28,669,939 surveyed sites. That's a 2.3% gain. Microsoft's IIS also gained .89%, achieving 20.64%. Sun/Netscape's iPlanet beat even with a .03% gain, for a 6.27% share. The rest, in total, were down.

Here are some of the improvements in the Apache 2 beta:

- Runs in a hybrid multiprocess, multithreaded mode.
- New Apache Portable Runtime and multiprocessing modules.
- Filtered input/output modules.
- IPv6 support.

The Apache Software Foundation is at apache.org.

Netcraft also reported that Compaq and AltaVista have followed Amazon's lead by moving its servers to Linux. Both were on Tru64 (formerly Digital UNIX, which was bought by Compaq along with the rest of Digital Equipment Corp). Compaq moved off Tru64 to Windows in January 2001, before moving to Linux. Netcraft is at http://www.netcraft.com/.

—Doc Searls

Stop the Presses: Linux on the PDA Agenda (and Vice Versa)

It's starting to look like a tsunami of Linux-based PDAs is about to spread out of Asia. From Japan, Sharp recently announced that it would roll out a new PDA based on Linux rather than an OS from Palm or Microsoft. The Korea-based G.Mate Yopy is a PDA that uses a speech interface from Conversay, a company headquartered in Redmond, Washington. Ericsson Singapore and Singapore's Centre for Wireless Communications have announced a jointly-developed "handheld computer" called the DelphiPad that runs Linux, features a 10-inch touch screen and is scheduled to sell in the fourth quarter of 2001 for under $1,000 US. VTech has the Helio. And you can put several forms of embedded Linux into Compaq's iPAQ and other PDAs.

But the PDA with major momentum at the moment, judging from the sudden upswelling of buzz in the Linux community, is the Agenda VR, from Agenda Computing. While Agenda is owned by Kessell International of Hong Kong, which also handles manufacturing, the company's whole agenda (pun intended) seems to originate out of its Irvine, California offices, where the company is run by its president, Bradley La Ronde.

Recently I was on The Linux Show with Brad, who seemed to be at least as committed to mobilized Linux as the OS' famous creator. I got the distinct impression that Agenda is a harbinger of change in the consumer electronics business, from one controlled by corporate giants to one controlled by small developers who take advantage of freely available technologies that are constantly improved by their surrounding development communities. I later found out that this particular show was one of the most popular in the history of the program.

Then a couple days ago I got this unsolicited e-mail:

> Went to my first linux users group meeting in like a year last night (http://www.nblug.org/, North Bay Linux Users Group) and the CEO/President/Developer from Agenda Computing was there giving a demonstration and talk about the VR3 Linux-based PDA's they're putting out....Don't know if you've checked them out before, but they're actually a lot cooler and more usable than I thought they would be.

Later he added,

> They're a little slow—but a big part of the discussion revolved around various ways of solving that. It was very, very cool to have a realistic, technical discussion with a CEO about their product. I spoke with him afterwards, and we agreed on some points where they're going to have difficulty in the marketplace, but the part I cared most about was his honesty. Very clued, in my view.

And that was just one guy. Agenda is clearly making some smart moves with the Linux community. From the start, Agenda's agenda has been to put out an inexpensive ($249 MSRP) basic PDA that runs Linux in a form so inviting to Linux hackers that they'll jump in and write all kinds of stuff for it. This appears to be exactly what's happening. There are a growing pile of independent developer sites, which, together with Agenda, have put together a rapidly growing pile of apps for the Agenda VR—and, presumably, for other LinuxVR-based (linux-vr.org) devices, which also include PDAs from Vadem, Casio and Everex.

In the words of Ian LeWinter, Agenda's VP Marketing, the Agenda VR will compete with Palm, Handspring and other PDA companies for a reason that has nothing to do with Linux' hermit crab-like ability to run in almost anything. The whole look and feel of the device “screams cool”. It's truly palm-sized (4.5" x 3.0" x 0.8"), comes in three colors, runs on NEC's 66MHz 32-bit MIPS processor, with 8MB RAM + 16MB Flash Memory, both IrDA and its own peripheral ports. Audio, too.

According to the independent supermegamulti.com/agenda/ site, there were 93 Agenda VR3 programs in the on-line software depsitory. Those include 23 apps, 16 games and 22 utilities. By the time you read this the number will certainly be much higher.

A review of the Agenda VR is in the works for a future *Linux Journal*.

—Doc Searls

They Said It

The radio market is now clearly driven by greed and corruption rather than creativity and talent. Something must be done to bring attention to this, and I strongly believe that swift federal action is necessary.

—Sen. John McCain, on record companies paying stations to play their music.

Greed is never good.

—Linus Torvalds

As far as the laws of mathematics refer to reality, they are not certain; and as far as they are certain, they do not refer to reality.

—Albert Einstein

Sex is the mathematics urge sublimated.

—M. C. Reed

2,500 statue miles is .0134 light seconds. If we take 10 gigabits/sec (OC192C) as a gigabyte/sec (rounding), we get 13.4 megabytes in flight at 2.5 gigabits/sec (OC48C), divide by 4 and get 3 megabytes in flight.

—Mike O'Dell

Windows has noticed you have changed your mind. Windows will reboot to recognize this change.

—Gates_throws_tantrum, on Slashdot

Irony is so dead.

—Don Marti

Human reality is socially constructed. That is, most of the “facts” that determine our daily lives are socially constructed facts, which are true as long as enough people believe them to be true. The right to own property, the right to not be murdered, indeed the right to continue to live at all; all of these are socially constructed rights, which are true only as long as enough of us believe in them.

—Rusty Foster

There's no matter on the Web and thus no distance. It is a purely social realm; all we have are one another and what we've written. And what we've written has been written for others. The Web is a public place that we've built by doing public things.

—David Weinberger

We hackers were actively aiming to create new kinds of conversations outside of traditional institutions. [The Net] wasn't an accidental byproduct of doing neat techie stuff; it was an explicit goal for many of us as far back as the 1970s. We intended this revolution.

—Eric Raymond

When you've commodity chips strapped together with commodity drives hooked together with fast Ethernet interconnects, then you want a commmodity OS, and Linux is it.

—John K. Thompson

Humans are destined to be party animals, and the technology will follow.

—Linus Torvalds

Archive Index Issue Table of Contents

Advanced search

Best of Technical Support

Various

Issue #87, July 2001

Our experts answer your technical questions.

Tar over rsh

I'm running a small private LAN with Linux Red Hat 7.0. The following command was entered on system “lucy”, with the resultant message:

```
[root@lucy]# tar cvf testbed:/home/someuser file.txt
Permission denied.
tar: testbed\:/home/someuser: Cannot open:
  Input/output error
tar: Error is not recoverable: exiting now
```

This output is from the /var/log/messages file on the system “testbed”:

```
Mar 30 08:14:57 testbed pam_rhosts_auth[853]:
  denied to root@lucy as root: access not allowed
Mar 30 08:14:57 testbed in.rshd[853]: rsh denied to
  root@lucy as root:
Permission denied.
Mar 30 08:14:57 testbed in.rshd[853]:
  rsh command was '/etc/rmt'
```

I have been reading about PAM but have not figured it out yet. Could someone give me assistance to make this command work? I don't mind if I temporally dummy up my security in /etc/pam.d, but my last attempt made it impossible to even log in. Ouch! —Les Hilliard, les.hilliard@home.com

On the machine where you want to put the tar file, create a .rhosts file in the home directory, with 0400 permissions. The .rhosts file should a single line, “X.X.X.X user”, where X.X.X.X is the IP address of the machine where the tar command is run and the user is the user id of the person running the command. Run the following command on the other machine:

```
tar -cvf root@mm:/root/aaa.tar work/
```

where mm is the remote machine name or IP address and work is the local directory. Beware that this will also allow rlogin without a prompt for password.
—Usman S. Ansari, uansari@yahoo.com

Error in Loading Shared Libraries

I recently installed Netscape 4.7.6 on my Linux machine (the machine was recently rebuilt, so the kernel and libraries are very recent). When I was trying to run it, I got the following error message:

```
/usr/local/bin/netscape
/usr/local/bin/netscape: error in loading
  shared libraries:
libstdc++-libc6.1-1.so.2: cannot open shared
  object file: No such file or directory
```

Checking my libraries confirmed that I didn't have that particular library installed, but a newer one:

```
# cd /usr/lib
# ls libstdc++*
libstdc++-3-libc6.1-2-2.10.0.a
libstdc++.a.2.10.0
libstdc++-3-libc6.1-2-2.10.0.so
libstdc++-libc6.1.so
libstdc++-libc6.1-2.a.3
libstdc++-libc6.1-2.so.3
```

I successfully resolved the problem by providing a symbolic link to the newer library:

```
# ln -s libstdc++-libc6.1-2.so.3
  libstdc++-libc6.1-1.so.2
```

This solved the problem, and Netscape is running smoothly. However, this incident left me with a number of questions:

What is the meaning of the .2 and .3 at the end of the filename?

Is this solution appropriate? I would have liked to create a shorter link, like libstdc++-libc6.1.so, but that didn't work. Would it have been more appropriate to actually find libc6.1-1 and install it next to the existing one?

My assumption is the Netscape binaries had the library version hardcoded (I believe this version of Netscape is not available as source code); is that correct?
—Michael, micky@alum.mit.edu

The reason a library maintainer changes a revision is there is a significant change in the underlying code or interface. The maintainer usually feels it would not be wise for a program dynamically linked with an older version of the library to automatically work with the newer version. “Appropriate” is in the eyes of the beholder. It is possible that the dynamically linked program you are

“tricking” could disastrously crash, destroying itself and other things. Most likely, though, it won't. But, it definitely would be safer to find the actual dynamically linked library. The Netscape 4.x binaries have some interfaces (to the dynamically linked library) and versions hard coded. —Christopher Wingert, cwingert@qualcomm.com

Mandrake without X

I am currently using Mandrake 7.2 as a server platform. I do not want to run X. I have tried all configuration options on the Mandrake install, and even manually deselected X components, but the install just goes ahead and installs X anyway. Is there a way of stopping X and X components from being installed?

Also, when in console mode, is there a way to stop the monitor from going into power save mode? I have disabled the apm dæmon and power management in the BIOS, but the monitor keeps shutting down. —Gerard Nicol, gerard.nicol@tapems.com.au

Use **rpm -qa** to uninstall packages you may not need. You should also keep a list of the packages you uninstall so you can install them again, if needed. To disable power saving in the console, do **setterm -blank 0**. —Usman S. Ansari, uansari@yahoo.com

How Can I Set the FTP Welcome Message?

I want to be able to change both the initial message displayed when a user opens an FTP connection to my system and the login message. I know this is supposed to happen in ftpaccess, but I can't for the life of me find the files referred to in the configuration files /welcome.msg and .message. Is it that these files simply don't exist, and the messages displaying are defaults? Help. —Jon Dewey, jmdewey@clunet.edu

These files exist in your anonymous **ftp** area. On Red Hat this is usually /home/ftp. If you place a text file called welcome.msg in that directory, it will appear when someone anonymously logs into your machine. —Christopher Wingert, cwingert@qualcomm.com

Wiping a Hard Drive MBR

I can't install Linux due to an MBR problem and the message **RAMDISK: Compressed file at block 0**.

I have tried to **cfdisk** but no luck. I tried all sorts of boot/rescue diskettes without luck for two years now.

How can I wipe this hard disk clean before I install RHLinux? —Joseph Lalingo, joseph.lalingo@ablelink.org

You can wipe the MBR with **lilo -u**. —Christopher Wingert, cwingert@qualcomm.com

Kernel Panic on Boot

I installed Red Hat 7.0 as a dual boot with Windows 98 on my Toshiba laptop. Now when I boot, I am unable to get to either operating system, just a series of bracketed numbers, and the following message:

```
Code:89 02 85 c0 74 03 89 50 04 b8 01 00 00 00 eb
  03 90 31 c0 c7
Aiee, killing interrupt handler
Kernel panic: Attempted to kill the idle task!
In interrupt handler - not syncing
```

I am unable now to boot from diskette or CD-ROM. —Neil O'Connor, bowstn@yahoo.com

Boot your laptop using the rescue floppy created at install time, and run **/sbin/lilo**. This will reinstall LILO, and you should be able to boot from the hard disk again. —Usman S. Ansari, uansari@yahoo.com

Archive Index Issue Table of Contents

Advanced search

New Products

Heather Mead

Issue #87, July 2001

DupLinux from Arco, Firewall-in-a-Box, Max Server Pages and more.

DupLinux from Arco

During the LinuxWorld Expo in Tokyo in May 2001, Arco Computer Products introduced DupLinux, an IDE RAID controller utility with background rebuilding and hot-pluggable drive capabilities. Designed to work with Arco's DupliDisk II IDE RAID 1 controller, users can configure, administer and rebuild their RAID array from the command line or through the X Window System. DupLinux also enables hot-plugging for hard drives.

Contact: Arco Computer Products, Inc., 3100 North 29th Court, Second Floor, Hollywood, Florida 33020, 800-458-1666 (toll-free sales), sales@arcoide.com, http://www.arcoide.com/.

Firewall-in-a-Box

Firewall-in-a-Box (FIB) is now available from EMAC, Inc. FIB is a fanless, small-footprint firewall that can securely share connections among several workstations or servers. Based on a customized Linux distribution, FIB controls data flow over an optional analog modem, cable modem or digital subscriber line. It is able to assign IP addresses dynamically to client machines (DHCP) and provides a caching DNS. FIB utilizes a low-voltage NS Geode GXLV-200 processor and a 50-watt power supply. The menu-based configuration utility is accessible via terminal or Telnet.

Contact: EMAC, Inc., 2390 EMAC Way, Carbondale, Illinois 62901, 618-529-4525, info@emacinc.com, http://www.emacinc.com/.

Max Server Pages

PlugSys announced the availability of Max Server Pages (MSP), a database-oriented server-side scripting product for web servers. MSP uses Xbase commands and functions so developers can quickly access data stored in DBF files and SQL databases. SQL queries and updates are done with the PlugSys ODBC Connector. The results are blended with HTML and JavaScript. MSP installs as a self-contained engine providing dynamic compilation and caching with runtime evaluation capabilities. A free edition (MSP/FE) can be downloaded from the company's web site.

Contact: PlugSys International LLC, 1636 Graff Avenue, San Leandro, California 94577, 510-352-2228, http://www.plugsys.com/.

Heroix eQ Management Suite

The new Heroix eQ Management Suite, from Heroix Corporation, is infrastructure cross-platform management software that allows monitoring tens to thousands of systems. eQ capabilities include detection, notification and resolution of application, system and network problems. A purpose-built rule engine powers system monitoring and is designed to emulate human knowledge and reasoning. The eQ suite features application autodiscovery, the Express Wizard interface, emergency repair and event reporting. eQ runs on a variety of Linux, UNIX and Windows platforms.

Contact: Heroix Corporation, 120 Wells Avenue, Newton, Massachusetts 02459, 800-229-6500 (toll-free), http://www.heroix.com/.

The Pockey

Pockey Drives released a new portable storage product called the Pockey, a USB external hard disk drive that is small, fast, portable and requires no additional power supply. The palm-sized Pockey is available in 10GB and 20GB capacities and is compatible with all laptop and desktop models. A USB cable connects the Pockey to a computer that transmits all data and power. The Pockey can be hot-swapped between computers without rebooting so file sharing is easy. The dimensions are 5" x 3 ½", and the transfer rate is up to 1.5MBps.

Contact: Pockey Drives, 21356 Nordhoff Street, Suite 109, Chatsworth, California 91311, 818-717-9556, info@pockeydrives.com, http://www.pockeydrives.com/.

VelociGenX

VelociGenX, from VelociGen, is a web services application development and runtime platform that uses XML technology to provide cross-platform connectivity. Companies are able to develop reusable web services that incorporate data from any source or application. Therefore, VelociGenX can wrap, link and run various XML components as meta-applications from both legacy and dynamic data sources. The meta-applications can then be run and results sent to a browser, e-mail, PDA, pager or cell phone.

Contact: VelociGen, Inc. 8380 Miramar Mall, Suite #105, San Diego, California 92121, 858-622-1164, info@velocigen.com, http://www.velocigen.com/.

PostgreSQL 7.1

The PostgreSQL Global Development Group has made PostgreSQL version 7.1 available for download from the web site and mirror sites. Key new features of the 7.1 release include: the write-ahead log (WAL), which increases data integrity and processing speed because only one modified log file must be flushed to disk; rows of any length, via the oversized attribute storage technique (TOAST); support for SQL92 outer joins; 64-bit C function manager support; and improved support and speed for complex queries.

Contact: PostgreSQL Global Development Group, P.O. Box 1648, Wolfville, Nova Scotia, Canada B0P 1X0, 877-542-0713 (toll-free), info@pgsql.com, http://www.postgresql.org/.

SANblade

QLogic Corporation introduced SANblade, a board-level platform that allows OEMs and ITs to standardize SAN connectivity products for server and storage systems. SANblade is designed to support a broad range of application-specific requirements with a common hardware interface platform, software management suite, driver interface and sales channel. Currently based on Fibre Channel products, the SANblade family aids design, acquisition and deployment of products around a standard CDI to speed up development and time to market.

Contact: QLogic Corporation, 26600 Laguna Hills Drive, Aliso Viejo, California 92656, 800-662-4471 (toll-free), http://www.qlogic.com/.

Archive Index Issue Table of Contents

Advanced search

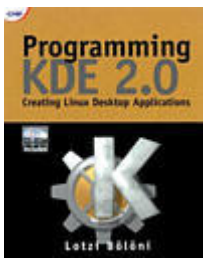
Programming KDE 2.0: Creating Linux Desktop Applications

Stephanie Black

Issue #87, July 2001

If you like KDE, this book is a fine piece of work and will undoubtedly speed you on your way to creating a “killer app” of the finest kind.

- Author: Lotzi Bölöni
- Publisher: CMP Books
- URL: http://www.cmpmedia.com/
- Price: $39.95 US
- ISBN: 1-929629-13-3
- Reviewer: Stephanie Black



Good programming books are hard to find. By “good”, I mean that things like an author's bias does not overshadow the value of information in the work, the writing is clear, appendices are used as addenda to the written work (and are not half of the text) and assumptions about user knowledge/skill levels are consistent and addressed in some kind of chronological order. Lastly, it's most helpful if the book engages—not enrages—the reader.

Programming KDE 2.0: Creating Linux Desktop Applications (CMP, 2001) introduces the reader to development of KDE applications. The approach Bölöni takes assumes at least a basic knowledge of C++ and systematically

takes the user through the basics of the Qt libraries, use of components and all the way to “expert touches” to make your new KDE application sparkle.

If you like KDE, this book is a fine piece of work and will undoubtedly speed you on your way to creating a “killer app” of the finest kind. You will have spent your money wisely.

If you do not like KDE, if you have found it wanting (especially its meager memory resources), you will probably not appreciate Bölöni's “boosterism”, his detailed expiation on the history of graphical desktops (unified graphical desktops in particular) or his utter refusal to broach the subject of memory management in KDE applications. The only part you might find a bit amusing is the opening statement of Chapter 1: “The K Desktop Environment (KDE) is the most popular desktop environment for UNIX-like systems and probably the largest open-source project ever undertaken.”

If you take things like this seriously, the author has completed his unofficial (though not unofficious) task of alienating his readers. If you take these things with a grain of salt, it is almost possible to enjoy—and certainly benefit from—the information within the book. Almost.

(For the record, the largest open-source project ever undertaken would be the GNU project, without which neither Linux nor KDE would have usable and free development tools to aid their development.)

Highlights

From a technical standpoint, there's not much to critique about Bölöni's work except its premise. The extensive overhead of C++, even without the KDE libraries, components, additional objects, etc. can be daunting for many Linux systems that are not running a minimum of 128MB of RAM. Such information is ignored, which I find a bit questionable.

Assuming the developer/user has a hefty amount of RAM and is comfortable with C++, Bölöni's technical expertise is evident. The explanatory text and illustrative code samples complement each other well. For the most part, no obvious glaring errors appear in either, until we get to the end of the book.

In Chapter 8: “Expert Touches”, Bölöni provides a wonderful discussion of communications in KDE, what protocols are involved, what classes actually do—things like addressing, “marshaling” data and registering an application. This is one section where the author provides some useful information. For example, his listing of code for “LocalChat” illustrates his points well. Although the code is lengthy, it is well-commented and is a real education in the grunt work of modern communications tools.

Another feat is from the “smart coders dupe stupid users” school of programming, by way of a flagrant attempt to explain away the continental-drift speed for which KDE applications are known. The phenomenon in question is termed “perceived performance”, as in the benchmarks say one thing, but the user perceives it differently. Whose benchmarks is he talking about? Which user(s)? Linux users? Or the subset we can only call “GNU-bies”? (Thanks to Jon Pennington for that wonderful term!) The ensuing discussion, on “tricking” the user by employing splash screens (to “hide” the length of time the application takes to load) is, one would have thought, beneath someone of Bölöni's caliber.

The author deigns to give us some “parting thoughts” in Chapter 9 that are either (or both) vague (“How to Make Money with KDE” says nothing about how, only that you can) or hypocritical (Java's advantages are paid for with a massive performance loss? Pots, kettles, lend us your callings!) and include the following boast:

> All the reasons I mentioned for using KDE for custom applications hold for off-the-shelf software, too. But you must consider one more thing: the size of the market. Given that KDE is bundled with all the commercial Linux distributions and with all the major Unices, when writing a KDE application, you are targeting practically 100% of the Linux/UNIX world.

Please. There are a fairly large number of GNOME users that would choose Glade in a heartbeat over KDE, to say nothing of some very competent and critical coders using Enlightenment, WindowMaker and even Blackbox. Bölöni, by not taking these users/developers into account, misses a large portion of his potential audience.

Technical Problems

It's a wonderful thing when a book about software includes the software to which it refers. The author has tried to do this and with some forethought as to the various distributions on which KDE 2.0 might be loaded. Having run previous versions of KDE, I wasn't squeamish about testing it. This was, after all, a newer release that is better, from some reports, than previous versions. (Some outfits insist that “eats more RAM” is synonymous with “better”.)

Suffice it to say that a certain libmng was missing from the Debian binaries, resulting in a grumpy reviewer, an unhappy 64MB of RAM (which was slowed down to the point of unusability) and a crashing bore of a hard drive. Looking for the missing library entailed a wild goose chase of several hours and bore no fruit. libmng requires zlib. Requires it but can't see copies of it.

Sigh.

Conclusion

If you have a lot of RAM, like C++ and like creating desktop applications, *Programming KDE 2.0* is quite an acceptable guide. If you're running Debian, don't use the CD that comes with the book; go to the KDE site (http://www.kde.org/) and download it. The book will come in handy as a reference. And if you don't like KDE, the title alone should put a stop to any impulse buying.

Stephanie Black is a writer—of words and code. When not writing, she runs a Linux consultancy, Coastal Den Computing, in Vancouver, BC Canada. In her off-hours, she's usually playing fetch with her cats or collaborating/colluding with her partner, a fabric artist and business manager.

Archive Index Issue Table of Contents

Advanced search