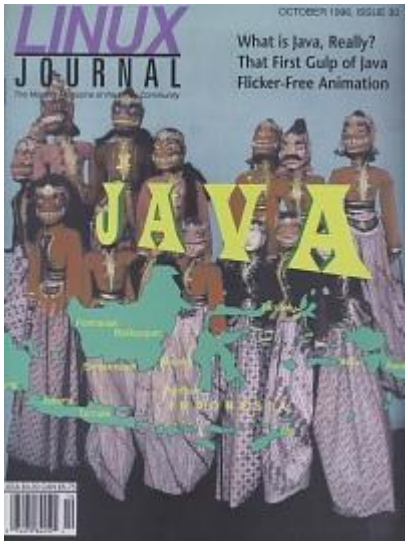
# *Linux Journal* Issue #30/October 1996




### *Features*

What is Java, Really? *by Rudi Cilibrasi*
Let's Skip the hype. This article explains what Java is and points you to the right places if you want to dive in.

Flicker-free Animation Using Java *by Paul Buchheit*
Currently the most popular use of Java seems to be in building applets. This article shows you not only how to make an applet, but how to make it look good.

That First Gulp of Java *by Brian Christeson and John D. Mitchell*
A relatively new technology, Java has experienced phenomenal growth. Why? Read on.

### *News and Articles*

My Next Pentium Is A DEC Alpha *by Bryan W. Headley*
Is a DEC Alpha a solution if you want a really fast Linux system? Here is one person's experience that may help you decide.

DEC AXP Review *by Bryan Phillippe*
Faster than a speeding bullet, able to leap tall buildings ... it's Digital's AXP (aka Alpha) computer.

### *Columns*

Letters to the Editor
From the Publisher The Politics of Freedom
New Column Linux Means Business
Stop the Presses

*Directories & References*

# What is Java, Really?

**Rudi Cilibrasi**

Issue #30, October 1996

Let's skip the hype. This article explains what java is and points you to the right places if you want to dive in.

Java conjures up a variety of images for computer users everywhere. Some picture flying heads and cartoonish figures that wave from web pages. Others remember segmentation faults and hung Netscapes. Most people are left staring at the steaming cup-o'-joe icon, having no idea what it all really means. This article will discuss what exactly Java is, and where it fits in the big picture of computing.

The Java programming language is roughly similar to C++, Objective C and Smalltalk, with some features removed and a few features added:

- It is *procedural*, like C, as opposed to functional like Lisp.
- It is *object-oriented*, like C++. It supports classes, single class inheritance, multiple interface inheritance, access protection and exceptions. It does not support operator overloading or templates (though it may support template-like constructs in the future).
- All casts are dynamic and safe; it's impossible to cast an object to an inappropriate type.
- Everything in Java is class-oriented. There is no global scope, there are no global variables, and there are no explicit function pointers.
- Arrays of arbitrary dimension are supported and act like objects.
- It supports *threads* and provides *synchronization primitives*. Threads, sometimes called lightweight processes, allow several separate execution paths to run concurrently through a program. Synchronization primitives ensure that separate tasks are performed in the correct order.

- It supports *modules*, so it doesn't need a preprocessor. There are no **#include** directives as the equivalent functionality is built into the languagex), no macros, no **#define**, etc.
- It supports *garbage-collection*, and there is no support for explicit deallocation. This simplifies programming dramatically and eliminates *all* memory allocation errors.
- All objects implicitly use *reference semantics*, which roughly means *every object acts like a pointer, but used as if it were a value*. All base types (**int**, **char**, etc.) use value semantics.
- Java is *safe*, meaning it is impossible to: de-reference a pointer outside a segment; use memory that has been deallocated; or use an object as the wrong type. A Java program will never cause a segmentation fault (actually, the Java interpreter *does* have a few bugs that can result in an application crashing, but these are being fixed).

Java comes bundled with a standard library called the Java API (Application Program Interface) that provides:

- Basic file input and output
- Networking, particularly TCP/IP, UDP, and HTTP
- Basic dynamic container classes like Vector, Dictionary, etc.
- Graphics and GUI programming

Another important component of the Java system is the compiler. A Java compiler translates source files (*e.g.*, Foo.java) into one or more class files (*e.g.*, Foo.class). The **.class** files correspond roughly to the **.o** (or **.obj**) files in C, but there is one crucial difference: while **.o** files contain machine-specific instructions, **.class** files contain only a generic pseudo-assembly-code called *Bytecode* that is uniform across all platforms. To enable this, every platform that supports Java must have a Java runtime that will interpret and execute Java Bytecode.

The underlying Bytecode to which Java compiles is described under Sun's Java Virtual Machine (or JVM) specification (java.sun.com/doc/vmspec/html/vmspec-1.html) A Java runtime emulates a JVM, so that it can run the Java Bytecode contained within **.class** files. The JVM is a key component to Java's ubiquity; rather than require that every individual application support multiple platforms, developers need only write one version of a program, and are assured that it will run everywhere. The only program that needs to be ported across multiple platforms is the Java runtime. Despite Java's youth, most major platforms are supported.

Let's now take a moment to trace the steps involved in letting you see waving animated characters within your web browser.

- A programmer on the Web somewhere writes a Java program, and converts **.java** source files into **.class** files using a java compiler. This programmer puts a reference to these **.class** files within an <APPLET> tag in her web page.
- You point your browser to her web page, and your browser sees the <APPLET> tag pointing to the **.class** files. Your browser then downloads her **.class** files over the net.
- Your browser starts up its Java runtime subsystem and begins executing the **.class** files that it downloaded.

At this point, you might be wondering what prevents a malicious programmer from doing horrible things to your computer while the comforting cartoonish character distracts you from the sound of your hard drive as it's being reformatted. The answer lies in the Java SecurityManager. This class, a part of the standard, implements a *security policy* for all untrusted programs. Here, untrusted programs are those that are downloaded over the net. In the current release of Netscape, the SecurityManager that the Java runtime uses does not allow untrusted Java programs to read or write files, nor does it allow them to make network connections anywhere except to the host from which the program came. This, then, forms a sort of sand box in which the Java program is confined.

Another key facet of the Java security model is the Bytecode Verifier. The Bytecode Verifier checks all incoming programs to make sure that they satisfy some basic safety constraints (*e.g.*, every pointer cast is a safe one, there can be no stack overflows, etc.). This prevents malicious programmers from creating bad **.class** files by programming Java Bytecode by hand in an effort to circumvent the safety features of the Java language. If your Netscape downloads a **.class** file that fails to pass the verification stage, it will simply refuse to run the code.

Most people view Java as *the* programming language of the Web. Though Java certainly serves well in this regard, it is actually much more general; it is appropriate for stand-alone, non-Web programming as well. You can write Java programs that can be run outside of a web browser, and as trusted programs they can do everything a C or C++ program might do, such as access the file-system. In fact, in the Linux development pre-2.0 kernels there is support for the Java binary format. This support means that you can run Java Bytecode programs just as you would run **ELF**- or **a.out**-format executables from the command line, without explicitly invoking a Java runtime. The great advantage

of writing programs in Java is that they are immediately portable across a wide variety of platforms and architectures.

One problem with current Java implementations is speed; performance is many times slower than C under most current implementations. This is because the Java Bytecode is interpreted, unlike native assembly code compiled from C programs which is run directly by the processor. The speed problem is being mitigated, however, by the development of so-called "Just-In-Time" compilers. These compilers (which are actually part of the Java runtime system) compile **.class** files into native assembly code on the fly. In so doing, they can speed up Java programs many times. Furthermore, since they only compile classes as they execute, the typical initial delay associated with traditional compilation is eliminated. As these compilers mature, we can expect the performance hit associated with Java to disappear.

At this point, you may want to know how you can become *Java-enabled*. If you simply want to run Java applets on your machine, (that is, Java programs transferred over the Web) you can do so using the latest Netscape beta, available from www.netscape.com/. If you'd like to begin experimenting with programming in Java, or running non-browser stand-alone Java programs, you can do so with the free Linux port of the Java Development Kit (JDK). Point your browser at java.blackdown.org/java-linux/Information.html to find out where to get this. If you'd like an alternative to the Java compiler included in the JDK, there is EspressoGrinder, a drop-in replacement for the JDK compiler. EspressoGrinder is written in Java, and thus can be run anywhere that a Java runtime has been ported. You can get it at wwwipd.ira.uka.de/~espresso/. If you'd like more information on Java, the JDK, and general Java happenings, check out Sun's site at: www.javasoft.com/java.sun.com/devcorner.html. A Java FAQ is available from sunsite.unc.edu/javafaq/javafaq.html.

The political and economic significance of Java is enormous, especially for Linux developers. Because of the high degree of virtualization in the JVM, developers are free to create their applications on platforms other than those that they target. Thus, it is now possible to use your Linux system to develop Java applications that can effortlessly tap into the popularity of Windows, Macintosh, HP-UX or SGI machines, to name a few. Furthermore, Linux will undoubtedly be able to reap the benefits of applications written under other platforms for exactly the same reasons. Overall, we can expect Java to usher in a new era of software interoperability, empowering developers and users alike on all platforms.

**Rudi Cilibrasi** (cilibrar@ugcs.caltech.edu) has been programming computers for over a decade; he works as a software consultant, specializing in Java, C++, and Linux.

# Flicker-free Animation Using Java

**Paul Buchheit**

Issue #30, October 1996

Currently the most popular use of Java seems to be in building Applets. In this article Paul shows you not only how to make a applet, but how to make it look good.

If you were awake this past year then you've probably heard of Java. If you were browsing the World Wide Web with a recent version of Netscape then you've probably even seen a Java applet or two. A *Java applet* is a Java program that interfaces with and endures the security restrictions of a web browser. Animated displays and games were some of the earliest and most popular uses for applets, although certainly not their only uses. You may have already noticed that, despite the popularity of Java animation, many applets don't do animation very well.

In order to illustrate beginning Java programming and animation techniques, I wrote a very simple slot machine in Java, its only moving part a lone, spinning wheel. To start, I will present a slot machine that uses one of the simplest and (unfortunately) most popular methods of animation. This slot machine won't look very good, but making it look better is not very difficult, and I'll present several methods for improving it. The source, images and actual incarnations of the slot machines described in this article are available at k2.cwru.edu/~ptb/lslot/. I recommend that you experiment with them as you read this.

This slot machine is composed of two images. The first image--the body of the slot machine—is really just a decorative so I have chosen our good friend and potential Linux mascot, Tux. The second image is really a strip of images that forms the face of the slot machine's wheel (Figure 2). We could load each of these images separately but that would probably slow down the loading process and complicate the code. These images are loaded inside the **init()** method through a call to **getImage()**, a method defined in the superclass **Applet**.

If you know C++ (or Java) you might notice that **init()** functions a lot like a constructor, i.e. it is used to initialize variables for a newly created object. There is a big difference however, in that a constructor is called when the class is instantiated, **init()** is called when the applet's host is ready to initialize the applet. Moving the **getImage()** code discussed above from **init()** to a constructor could cause it to break.

When the applet's host finally decides to display the applet it will call **paint()**. Looking at **paint()** you should notice that it gets passed a variable **g** of type **Graphics**, it is with this variable **g** that you will do your drawing. In fact, all the drawing that an applet ever does is through some instance of **Graphics**. The first line of **paint()**,

**g.drawImage(body, 0, 0, this);**

will draw the image **body** at position 0, 0. The fourth argument to **drawImage()**, **this**, specifies an **ImageObserver**, **this** is simply a reference to the current object.

An **ImageObserver** is an instance of any class that implements the java.awt.image.ImageObserver interface, meaning that it has been declared to implement **ImageObserver** and has a method **imageUpdate()**. In the following example, an instance of the class Foo would be a valid **ImageObserver**.

```
class Foo extends Object implements ImageObserver
{        ...
        public boolean imageUpdate(Image img, int flags,
          int x, int y, int w, int h) {
                ...
                return true;
        }
}
```

Interfaces provide a limited, yet safe and usable form of multiple inheritance. Fortunately, **java.awt.Component**, an ancestor of the superclass, **Applet**, is already an **ImageObserver** so no code needs to be written here.

If the image is not yet fully loaded the system will make note of the **ImageObserver**. Later on when more, but not necessarily all, of the data is ready, **imageUpdate()** will get called. By default, the **imageUpdate()** method for an applet will call **repaint()**, which causes the “visual loading” effect where the image gets painted as the data is loaded.

The next thing that **paint()** does is create a new **Graphics** specifically for drawing the wheel on the slot machine. This is accomplished by a call to **createForWheel()**, a method that I added. As mentioned, all drawing is done using a **Graphics**, and every **Graphics** has a rectangle somewhere on screen or in memory, to which it can draw. If a command such as **drawImage()** involves

drawing outside this rectangle, the part that extends outside the rectangle is clipped. This is very useful if we only want to display a portion of an image.

The **createForWheel()** method creates the new **Graphics** by a call to the **create()** method of the **Graphics** that was passed to **paint()**. Four integers are passed to **create()** which specify the location and size of the new **Graphics**'s rectangle—in this case, the position and size of the wheel. Now that we have this new **Graphics** we can do things such as:

```
drawImage(strip, 0, -55, this)
```

Even though **strip** is over 500 pixels tall and the specified coordinates put it in the upper left-hand corner, a nice 55 by 55 pixel square (the size specified when creating the **Graphics**) shows up at the position specified for the wheel that displays rows 55 through 110 of the image **strip**.

Once the new Graphics is no longer needed, its **dispose()** method should be called. This probably seems a little odd since Java has automatic garbage collection (meaning that memory does not have to be explicitly freed as in C or C++). The reason for calling **dispose()** anyway is that Garbage collection is not immediate and the **Graphics** could be in possession of limited system resources.

This slot machine, like most, does not spin its wheel all the time. In fact, its wheel only spins in response to a mouse button click and then only for a short time. Looking at the **mouseDown()** method you can see that it simply creates a new **Thread** called “spinning”. When a thread is started it calls the **run()** method of the **Runnable** (another interface) object specified in the thread's constructor. In this case the object is this applet, **this**.

The slot machine's exciting spinning action is implemented in **run()**. The first thing that **run()** does is to ask **getNewItem()** where it's going. The **getNewItem()** method just returns a random number from 0 to 5 specifying the stopping item on the wheel. The **run()** method then calculates how far, in pixels, the wheel must travel to get there, including the number of items that should spin past the front before the wheel stops. After this **run()** simply loops until the wheel reaches its destination. Calculate the new position, repaint, sleep, repeat. Once finished **run()** just sets **spinning** to **null** so that clicking the mouse button again starts the wheel spinning, and **run()** returns.

Try the applet! Looks awful, doesn't it? Probably the most obvious defect is the ugly grey flashing that appears while the wheel is spinning. Fortunately, this problem is very easy to fix—every time **repaint()** is called the system asynchronously calls **update()**. **update()** is an inherited method (defined all the way back in the class java.awt.Component) that draws a rectangle the size of

the applet using the background color and then calls **paint()**. This rectangle draw is the source of most of the flicker—since we are going to immediately draw over the whole region of the applet it is not only annoying but unnecessary. To fix the flicker, just insert the following method in the space between **createForWheel()** and the **paint** methods (see Listing 1):

```
public void update(Graphics g) {
        paint(g); }
```

Now run the applet—doesn't it look much better? However, if you look closely you might notice that the wheel has a slight black flicker, and the animation is a little rough. This problem is similar to the previous one—the body of the slot machine, which just has a black square where the wheel should be, gets drawn before the wheel graphic. so for an instant there isn't any wheel. One popular solution for this problem is double buffering: using an off-screen buffer to hold the image while it is being drawn. Now the part of the body hiding behind the wheel will never appear.

To add double buffering to our applet, the first thing that you must do is create a buffer, called (appropriately enough) **buffer**. Next add an **Image** instance variable called **buffer** into the class and insert into the **init()** method the line:

```
buffer = createImage(size().width, size().height);
```

Now **paint()** must be modified so that it will first draw into the buffer and then draw the buffer itself onto the screen. This new **paint()** should look something like this:

```
public void paint(Graphics g) {
       Graphics bufG = buffer.getGraphics();
       bufG.drawImage(body, 0, 0, this);
       Graphics clipG = createForWheel(bufG);
       drawWheel(clipG, currentWheelPos);
       clipG.dispose();

       g.drawImage(buffer, 0, 0, this);
}
```

Now run the applet—looks much nicer, no more flicker! But it seems awfully inefficient to redraw the entire area of the applet window when only a small part is changing, doesn't it? Another easy fix! Simply change the line:

```
repaint();
```

found in **run()** into:

```
repaint(wheelPosX, wheelPosY, wheelSize, wheelSize);
```

This new call tells the AWT system to update only the specified rectangle and leave the rest of the window alone.

It seems that, as before, we are drawing the body of the slot machine more often than needed, only this time it's to the buffer instead of the screen. It's possible to work up a scheme where only the wheel gets repainted to the buffer, fixing this complaint, but, as you may have already realized, there is a better way.

This buffer is silly, not only is it starting to get complicated but having some big **Image** in memory is a big waste, especially for more complex applets (such as a big, complex, slot machine). Buffer images have their place but buffering the whole applet is rarely a good idea. Why not just forget about using repaint and instead draw the spinning wheel right inside of **run()**? Good idea. Going back to the code that we had before adding in that buffer, modify **run()** as follows:

```
public void run() {
        // Gets something to spin to.
        int nextItem = getNewItem();
        int pos = currentWheelPos;
        int finalPos = (itemsToSpin + nextItem) *
                wheelSize;
        Graphics g = createForWheel(getGraphics());
        while((spinning != null) && (pos != finalPos))
      {
                pos = findNextPos(pos, finalPos);
                currentWheelPos = pos % stripLen;
                drawWheel(g, currentWheelPos);
                getToolkit().sync();
                try {
                        Thread.sleep(delay);
                } catch(InterruptedException e) { }
        }
        g.dispose();
        spinning = null;
}
```

Now we are simply getting the **Graphics** that we need to draw the wheel and calling **drawWheel()** directly every time we move the wheel. The trick here is to call

```
getToolkit().sync();
```

when we want the drawing to appear. Without the call to **sync()** the system would wait until several drawing requests arrive, resulting in jumpy animation.

Finally the slot machine is finished! I think you can see that although the code is almost identical to that of the first slot machine, the resulting animation is much smoother.

For more complex animations you will probably want to use a combination of the methods presented here. For example, let's say you want to create a box with two balls bouncing around inside. In most cases you will simply have a **drawImage()** for each ball, but what happens if the two draws overlap? You may end up with flicker in the area of intersection. One solution would be to double buffer the draws whenever the two images intersect.

As usual, if you are having a hard time with some aspect of your applet you can (hopefully) find an applet that does something similar and look at the source. The largest collection of Java applets can be found at Gamelan (http://www.gamelan.com/), you should also find a link to the winners of the Java Cup International there. Another good place to find applets is the Java Applet Rating Service (JARS) at www.jars.com/. As the name implies, the Java Applet Rating Service rates applets based on a number of factors including quality of code (if the code is freely available).

**Paul Buchheit** (ptb@po.cwru.edu) is an inmate at Case Western Reserve University. When he's not busy sleeping, he's awake.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL



# That First Gulp of Java

**Brian Christeson**

**John D Mitchell**

Issue #30, October 1996

A relatively new technology, Java has experienced phenomenal growth. Why? Read on.

When we told a newbie friend that we had just co-authored a book on Java, he said, “You may be surprised to learn that I've heard of Java.” We weren't. In little more than a year Java has gone from an obscure back-room project at Sun to an all but universal topic of interest and speculation in the computing community—and we would be astonished if it hadn't.

If Java were just another powerful, expressive, object-oriented programming language, a few developers would have glanced at it, said, “that's nice.” and gone back to plowing their way through C++ code. If it merely offered a far more flexible, secure, and transparent way to enrich web page content with small programs, a few web surfers would have gotten about as excited as they did over “plug-ins”. If it only provided a more convenient way to distribute large applications across heterogeneous platforms, a few large companies would have begun investigating its potential.

Java combines all these features and many more, however, and it is this extraordinary combination that uniquely qualifies Java to capture the interest of an entire industry, and to demand significant investment by most of the big names in that industry: Borland, Intel, Microsoft, Novell, Oracle, SGI, and Symantec among many others. Even a fairly brief technical overview should suffice to explain the unprecedented enthusiasm that has greeted Java.

## Language Features

Like C++, Java capitalizes on the popularity of the C language, and preserves much of its compact, expressive character. Unlike C++, it makes no attempt to

maintain backward compatibility and, for that reason and others, it has many advantages over its currently popular cousin.

While C++ provides syntax to achieve encapsulation, inheritance, polymorphism, and other features of object-oriented programming, it also supports traditional structured (and unstructured) programming. Java does not. Every Java program we create, even the first “hello world,” is object-oriented, preventing us from backsliding into old, familiar—and less productive—ways.

The need to support two utterly different types of programming makes C++ needlessly complicated. For example, a parameter can be of either a built-in type or a class type, and may be passed by value, by reference, or via a pointer: six possible combinations. In Java, a built-in parameter is always passed by value, a class-type parameter is always passed by reference: just two combinations, and no decision required. Much simpler. Furthermore, stronger typing eliminates many of the implicit type conversions that introduce unexpected ambiguity.

“Wait! No pointers?” Yup. No pointers. C needed them to support array processing and a primitive form of call by reference. Java supports true call by reference (as C++ does) *and* implements arrays as a built-in type. So, no pointers. No uninitialized pointers. No forgot-to-check-for-NULL pointers. No tears-at-midnight-trying-to-find-the-mis-aimed pointers.

Global variables also meet a cruel but well deserved fate. In Java, *every* datum is neatly encapsulated, accessible only by operations of the class of which it's a member. Thus another major source of programming mistakes disappears entirely. Automatic garbage collection gets rid of yet another: the “memory leaks” that result when we fail to release memory allocated dynamically.

Java also discards a mechanism that “seemed like a good idea at the time”: the preprocessor.

Early C compilers avoided multiple parsing passes by supporting forward declarations of functions and data. Programmers could ensure consistency among declarations, definitions, and uses by placing the declarations in separate header files, then directing the preprocessor to “#include” them in source files. C and C++ implementations have continued this tradition into an age in which systems are so large that we must pre-compile the header files themselves to get acceptable rebuilding times—and header-file maintenance becomes a major task in itself.

In Java, class definitions are not split between header and source files. Simple “import” statements tell the compiler which classes are needed by the current

source file, and the compiler does the rest. Declaration order becomes unimportant. This scheme requires a bit more sophistication from the compiler, but one result of eliminating the preprocessor is that Java programs generally take less time to build than C++ programs. Another is that the developer's life becomes much easier.

The C/C++ preprocessor solves problems other than centralization of declarations, of course, but the Java compiler handles most of these other problems easily. And one other major problem disappears altogether:

The need to support multiple platforms obliges C and C++ developers to devote considerable effort to maintaining alternate blocks of code, and using conditional-compilation directives to ensure that only the correct blocks are compiled when they rebuild a particular system version.

Java makes all that tedium go away, because the language is independent of any particular machine architecture. Java programs are fully portable, not only the source code but the executable code as well, courtesy of the Java Virtual Machine (JVM).

### The Virtual Machine

Most modern languages are “fully compiled.” The compiler generates the “native code” of a specific target platform, i.e. the machine language appropriate to a particular operating system running in a particular processor. Once a program is installed on a user's machine, the operating system executes its instructions directly—an arrangement that achieves efficiency at the expense of portability. For example, if you are running Linux on a Pentium-based PC and creating a C program with GNU's gcc compiler, the resulting executable will run just fine on your machine and ones like it, but not on a Pentium running OS/2, and not on a DEC Alpha running Linux.

If you want to distribute your program widely, you will need to recompile it for a dismaying number of platforms, probably using a number of different development tools. Oh, and you want to keep supporting your software after sale, as well? Nice of you—start hiring. Experience has shown that the long-term effort of maintaining software products on multiple platforms far exceeds the effort of developing them in the first place. And the costs are proportional to the effort—better hunt up some heavy financing to pay all those people.

Java eliminates the complexity of cross-platform development and support through its reliance on a “virtual machine.”

As the word “virtual” implies, a Java compiler's target is a machine that does not actually exist. Instead of generating the native code of a particular platform, it

produces “bytecode”, a sequence of 8-bit codes that no actual machine can execute directly. Your program *will run*, however, and not only on your Linux box, but on any platform that supports Java—and these days that's the same as saying “on any popular platform.”

### The Run-time System

To execute a Java program, a machine must have a Java Run-time System (JRTS), an implementation of the JVM for that platform—but that is *all* it needs to run *any* program written in Java. The JRTS executes the bytecode much as an operating system executes native machine code. Because the run-time system handles all those nasty machine-specific issues for every program, the program itself does not have to.

It is a common mistake to confuse the run-time system with the virtual machine. Even people who should know better sometimes refer to “a program running on [a particular computer's] virtual machine”—and thereby conceal a crucial distinction. Part of Java's unique character is that only one piece of software, the JRTS, knows anything about the particular platform. Programs themselves remain blissfully ignorant of hardware dependencies—and so do programmers. They write their code for a machine that does not exist, serenely confident that doing so makes it portable to any popular platform.

A JRTS loads compiled classes as needed, performs security checks, and dynamically binds calls to methods. At that point many run-time systems begin executing the bytecode, interpreting each one as it is encountered. This continuous interpretation limits execution speed, and is the source of many early complaints about poor performance. An increasing number of Java implementations solve this problem by performing a second compilation step, “just in time.”

### Just-in-Time Compilation

Native-code compilers produce fast executables at the expense of portability. Java compilers that produce bytecode achieve portability at the expense of speed—*if* the JRTS interprets each instruction every time it is encountered.

Many run-time systems don't. In place of the interpreter they include a just-in-time (JIT) compiler. The first time the JRTS loads a portion of bytecode, the JIT compiler translates it into native code. Thereafter, the run-time system executes the native code instead of interpreting the bytecode; execution speed improves dramatically.

It is worth stressing that users get the speed of fully compiled programs *without* sacrificing portability. The JIT compiler is part of the JRTS, *not* the Java

source-code compiler, so all platform-specific knowledge resides only in the run-time system, on the user's machine, where it belongs. Software developers continue to compile and distribute the same portable, architecture-neutral bytecode files.

A second compilation step is not as expensive as it might sound. JIT compilation is actually quite fast in practice, because the most time-consuming tasks are completed in the first translation, from original Java source code to bytecode. JIT-compiled code is currently running 20 to 30 times faster than interpreted bytecode; this level of performance compares favorably with that of object-oriented code written in C++. Future improvements could boost this ratio to 50 or more, which would put Java executables on a par with optimized C code.

### Security Concerns

Java's virtual-machine concept improves security as well as portability, and at several levels.

Because a traditional fully compiled program is in native code, it is in an uncomfortably good position to exploit weaknesses in the operating system or hardware, and do serious damage. By contrast, Java bytecode is architecture-neutral, and what it does not know about the platform it cannot exploit. This “passive protection” is only the beginning, however.

Strong typing, including the addition of a boolean type, the replacement of pointers with type-safe references, and the elimination of other troublesome features makes it possible to perform run-time checks that validate a program's correctness.

In addition, the run-time system's Bytecode Verifier validates each program at load time, in several ways: It simply rejects any file that does not adhere to the distinctive bytecode-file format, thus avoiding execution of what might appear to be valid Java instructions but are not. When satisfied that a file is in the proper form, the verifier examines the bytecode itself for ill-formed constructs. It then goes on to search for errors usually not detected before run time, such as stack overflow.

Another part of the JRTS, the Class Loader, further enhances security by isolating classes from each other in separate security domains. To guard against malicious code, it separates classes that are built into the run-time system itself from classes local to the user's account, and separates both of these from classes that come from other users and other systems. An ill-intentioned “foreign class” thus cannot disguise itself as a more trusted class.

Users are understandably concerned that a virus or a Trojan Horse will enter their systems by way of an applet downloaded from the Internet. To guard users' systems, run-time systems employ combinations of security features Java makes possible, above and beyond bytecode verification and class partitioning. A Web browser or other package typically enables users to select from among multiple security levels, so that they may deny or limit “untrusted” applets' access to network connections and local file stores. Clearly visible marks distinguish windows created by trusted and untrusted applets so that the latter cannot masquerade as the former.

Much has been made of the risks inherent in downloading executable code over the notoriously insecure Internet. Experience with “plug-ins” has created some justified worry, but it is important to learn from Mark Twain's proverbial cat, and not shy away from a cool stovetop just because we once jumped onto a hot one. Java is too new for us to dismiss all such concerns blithely, but its many security features make it much safer than comparable technologies.

Some will not be satisfied with any risk level above zero; for them the only counsel can be complete abstinence from the pleasures of the Internet. Others realize that some risk is an inevitable feature of life in this world, and they can protect themselves by obtaining a Java Run-time System from a reliable vendor, through means as secure as those they use to acquire other software. Doing so should bring risks down to a level acceptable by most.

## Conclusion

The first uses of any new technology are often relentlessly trivial. If our only exposure to Java has been cutesy animations and downloaded calculators, it all too easy to underrate its potential. We hope this brief overview has shown that Java offers much more than bouncing heads—even though we didn't have the space to describe the neat way Java separates inheritance of implementation from inheritance of interface, and its built-in support for multi-threading, and....

**Brian Christeson** with John Mitchell, co-authored of *Making Sense of Java*. They are working on professional courses, other books, a compiler, and consulting/development projects related to Java, Tcl/Tk, and other languages. Brian lectures on OO analysis, design, and programming at major companies in the U.S. and abroad.

**John Mitchell** with Brian Christeson, *Making Sense of Java*. They are working on professional courses, other books, a compiler, and consulting/development projects related to Java, Tcl/Tk, and other languages. John developed PDA software in OO assembly language, and writes two columns for JavaWorld magazine.

# My Next Pentium Is A DEC Alpha

**Bryan W. Headley**

Issue #30, October 1996

Is a DEC Alpha a solution if you want a really fast Linux system? Here is one person's experience that may help you decide.

With the propagation of Linux onto non-Intel platforms, we are no longer tied to PCs for running our favorite *nix. I have been getting very leery of the “Pentium per 6 months” phenomena we've been seeing the last couple of years (i.e., whatever you buy is obsolete before you get it home). Each provides incremental enhancements, but nothing to get too excited about.

My goals were to find a new machine that could run Windows NT well, and provide me with an interesting platform for Linux work. (Well, someone has to study the enemy!) Limited to Pentiums, running NT on anything less than 133 MHz would not cut it. Frankly, I'm not expecting anything fun from the Pentium family until the HP/Intel PA-Risc merger chip is released (the forthcoming P7). So, it was off to other directions. A quick market survey indicated that DEC Alphas met my requirements, are competitively priced, and deliver good performance.

I settled on the DEC Universal Desktop Box (UDB), *a.k.a.* the Multia. This is a fairly compact (2.8x12.5x12.5 in) package which includes a 166- or 233-MHz Alpha, 24MB memory (expandable to 128MB), 256KB cache, Ethernet, floppy, two serial, one parallel, SCSI-2, sound and video components. The 166-MHz unit comes with an internal 340MB hard disk, or else a 520MB unit. Expansion is possible through a PCI card slot and two PCMCIA slots. The video chip supports multisync monitors from 640x480 to 1280x1024, 60 Hz to 75 Hz refresh rates; and the sound card is a clone of the Microsoft Sound system.

## Extra Goodies

I found that the internal hard disk was too small. Removing it, however, was out of the question. Digital put in a 2.5" form-factor drive, a size popular in laptops,

but which are generally IDE drives. I could force a 3.5" SCSI-2 drive in, but who would want the drive I removed? It's not too bad. I use the internal drive for boot partitions and swap space, and an external 1GB disk and a CD-ROM drive.

The Multia comes with a three-button PS/2 mouse, but no keyboard. You can obtain UDBs that support PS/2- or PC/AT-style keyboard connectors. Choose a keyboard type and a standard Multi-Sync monitor, and you are ready to run.

Alpha motherboards have PCI adaptors. Therefore you have a ready supply of PC-compatible adaptor cards that can be used with the Multia. For example, there are drivers for joystick adaptors. Even though the UDB comes with most of the features you would want anyway, there's no reason why you could not install a different video card, ISDN controller, etc., as drivers become available.

What makes the UDB special is the lack of frills. Digital, in an effort to keep the price down, markets the machine without software or licenses, and only about three pages of “documentation”--a pictorial essay on how to connect external devices to the connectors, how to open the case and insert RAM, a pointer to *Linux on Alpha* homepage (www.azstartnet.com/~axplinux) and information on how to join the *declinux* mailing list.

### Thanks for the Confusion

Digital decided that they rather liked the “Multia” name. The name therefore refers to both the Alpha version I am describing to you, as well as an Intel Pentium version. The boxes look very similar, and target the same market. I have not seen documentation that refers to the Intel Multia as a UDB, whereas the names are interchangeable for the Alpha version. Windows NT refers to the unit as a “Multia”, and the Linux FAQ prefers “UDB”. So take your pick, but be sure the DEC person knows which one you mean.

### Quick Introduction to Alpha Linux

This machine seems perfect for Linux and Windows NT. In fact, DEC, seeing a market for a low-cost *nix, participated in developing the port with Linus and the Internet community. The result is a 64-bit Unix that is binary-compatible with statically-linked DEC Unix (formerly OSF-1) applications. So, the availability of commercial software is not too big an issue. If you need a program like Netscape or WordPerfect, get the DEC Unix version. If it is statically linked, it should run with no problems.

Alpha Linux is interesting in that one of the main repositories of platform-specific code is kept on **gatekeeper.dec.com**. [See “Porting Linux to the DEC Alpha”, by Jim Paradis, in *LJ* #19—ED] Equally interesting is the fact that a lot of the platform-specific code is written (and source shared by) DEC engineers.

## Where Are We on the Kernel Source Tree?

The DEC Alpha source has been merged back into the generic Linux version 1.3.x. The result is, you can go to Linus' home page and download and build the linux-pre2.0.10 with full Alpha support. But, there is no support (yet) for sharable libraries (neither about QMAGIC/ZMAGIC, nor ELF). Plans are afoot to go to ELF while retaining DEC Unix compatibility via statically-linked binaries.

## Available Alpha Distributions

BLADE 0.3 based on 1.3 kernel; freely available

Red Hat 2.1 1.3.57 kernel; commercial distribution; shipping

Craftworks 2.1 1.3.89 kernel; commercial distribution; shipping soon

Of these, BLADE 0.3 is the original *stable* distribution. The README file at gatekeeper.dec.com suggests that you go with a commercial distribution instead. BLADE therefore is basically recent legacy code, albeit freely available.

BLADE, Red Hat, and Craftworks all come with the requisite kernel, compilers, libraries and utilities you would expect from a baseline Linux distribution. The commercial distributions, however, come with all the “goodies” you would expect (and get) from their Intel brethren—such as editors, networking, X-Windows, TCL/TK, etc.

What differentiates one commercial distribution from the other is the package mix that each vendor ships with his CDs. Ease of use of the installation packages has to be weighed in. The documentation is essentially the same for the Alpha and the Intel versions. Both Red Hat and Craftworks, however, come with installation instructions. Red Hat adds the Alpha FAQ to the documentation. Either way, there is little enough unique about the AXP to require custom documentation.

My UDB came pre-installed with Windows NT and Linux (I had ordered Red Hat 2.1). The vendor, DCG Computers, installed a pre-release of the Craftworks 2.1 package, so I could help them compare and contrast the two versions. Here are some of my findings.

## Distribution Comparison

The Craftworks product is newer than Red Hat 2.1; you get Linux kernel 1.3.89 versus Red Hat's 1.3.57. Both are fairly stable, and of course, you can always download the latest kernel no matter which package you use.

The DEC DC21030 display controller chip (*aka* the TGA) is built into the Multia. The Red Hat 2.1 does not come with an X display driver for this chip, although Craftworks does. Other video drivers come with the XFree86 port for the Alpha, including the S3, P9000, and Mach 64 (which both Red Hat and Craftworks have.) When Red Hat 2.1 was released, you had to buy a PCI video adaptor to run X on the UDP. In the interim, DEC has released an X server for the TGA, which Craftworks includes. Craftworks also comes with Lesstif, a Motif 1.2-like widget library.

The great equalizer is the installation program. Both run reasonably well, but each has little personality quirks that provide a few minutes' grief. The Red Hat package will partition your hard disk, make the swap partition (mkswap), but then forget to make the ext2 partition (mke2fs). Fortunately, you can switch over to the second virtual terminal and take care of this omission before continuing the installation process.

The Craftworks install package is somewhat overzealous in enforcing the size of the swap partition. They want you to have a 32 MB swap. That is fine, but I made a smaller one with Red Hat previously. The notification/irritation prompt keeps coming up about the small swap partition. It wouldn't let me do anything about it, such as kill it, and resize the partitions, but it was very persistent in its warnings. I finally had to bring up Red Hat's partition editor to delete all partitions by hand before I could continue. It could understand an empty partition table, but one that was made incorrectly beforehand could not be dealt with.

The trials and tribulations provided by the install programs yielded more amusement than anything else. So, the question is, which distribution would I use? Well, before I chicken out and tell you that I hand-merged things that I liked from both distributions (such as: glint from Red Hat, NYS shadow passwords from Craftworks, the Craftworks X Server, 1.3.89 kernel, etc.) let me mention to you that not all the candidates have been heard from.

Red Hat is scheduled to release version 3.0.3. I noticed that they have the new X Server. No doubt their *standard* packages have been updated to match their Intel 3.0.3 offerings.

### Real Trials and Tribulations

On PCs, the boot ROM attempts to read the initial sector of the primary floppy drive, or of the primary hard disk. Bootstrapping from that sector, the rest of the operating system is loaded. This design is fairly limited and the fact that the DEC moves away from that nonsense is the good news. The SRM Console Loader for the Multia comes with the ARC loader. ARC is pretty cool, insofar as

you can have multiple operating systems configured (from which you may choose at boot-time); these systems' boot sector can exist on any partition of any addressable device, including the floppy disk or CD-ROM. In fact, you can have vmlinux on the floppy drive and have the root partition be on a hard disk. This is all the good news.

The bad news is that I am a dangerous fellow. As I mentioned, I had DCG Computers install Windows NT and Linux together. Unfortunately, too much space was given to NT (more appropriately, I like to fit 2GB of Linux onto a 1GB partition). So, there I was, an hour after unpacking everything and firing it up, readjusting the partition table and trying to keep the ARC loader appraised of what I was doing. Well, I ended up with ARC knowing of NO operating system, and the partition table being empty, and I could boot neither NT nor Linux! Now comes the moment of gravely earned education...

Craftworks comes with the boot and root/installation partition on floppies. There are instructions on how to reprogram ARC to understand that the floppy disk has the operating system. Red Hat has the images of these floppies on the CD-ROM, but you have to find a way to get those downloaded to floppy. A friend with a PC and CD-ROM drive is useful (definitely, definitely, *definitely* make master copies of those floppies after you finish your installation).

ARC was written by Microsoft and DEC. Its specialty is loading operating systems from file systems it understands, specifically MSDOS FAT partitions. Once you point to one, you tell it which executable to run in order `to load the desired operating system. For Windows NT, that would be OSLOADER.EXE. For Linux, that would be MILO. But, the requirement here is that your loader has to be found on a FAT file system (it could be in any subdirectory, given whatever name you'd like). Let me add to the fun by telling you that I initially received the UDP without the Craftworks CD-ROM and boot diskettes (they came the next week). Oh, and don't forget the three pages of documentation: none of the pictures told me what I wanted to know about ARC.

Thankfully, I had Windows NT, and ARC was written to simplify NT support. There's a menu option called “install NT”, which will install from the CD-ROM. The Microsoft documentation for any non-Intel platform was sort of humorous, saying, “see the vendor's documentation for detailed installation instructions for your platform.” But, then it said to run **cd:*system*\setupldr**. System, system; what's a *system*?!?

Thankfully, the Pentium was nearby, and a quick directory of the NT's CD-ROM yielded that there were several directories, named after risc processors, where setupldr could be found. One of which is “alpha”... (Insert “D'Oh!” here).

Off to the races I went. Setupldr allowed NT to be installed, which gave me a nice FAT partition. On that Windows NT partition I placed a copy of MILO. A fun Saturday, which I stretched out to 4 am Sunday as the result ...

So, could I have moved the Red Hat CD to the Pentium and had it make the boot floppies? Yes, but if I remained Alpha-centric, I was doomed. Also, I had to learn ARC, and expected that having Windows NT install itself would educate me enough to get through it. Which it did; but if you look at something like this with no background, it looks scary:

```
LOADIDENTIFIER=Linux
SYSTEMPARTITION=multi(0)disk(0)fdisk(0)
OSLOADER=multi(0)disk(0)fdisk(0)\linload.exe
OSLOADPARTITION=multi(0)disk(0)fdisk(0)
OSLOADFILENAME=\milo
OSLOADOPTIONS=
```

Which, if I look at **SYSTEMPARTITION**, tells me that the system partition can be found off the Multia, off of the floppy disk controller, on floppy disk 0. As opposed to my current Linux setup:

```
LOADIDENTIFIER=Linux
SYSTEMPARTITION=scsi(0)disk(0)rdisk(0)partition(1)
OSLOADER=scsi(0)disk(0)rdisk(0)partition(1)\linload.exe
OSLOADPARTITION=scsi(0)disk(0)rdisk(0)partition(1)
OSLOADFILENAME=\udb.arc
OSLOADOPTIONS=boot sdb1:vmlinux.gz root=/dev/sdb1
```

Which says that I should go to the SCSI controller, a SCSI hard disk, raw hard disk 0, partition 1. Do you notice that this looks like a hierarchical file system? And so it does, except that it is describing the controller/hardware path to get to the boot code.

### After the Storm the Calm

Of course, I caused my own disaster, but that makes the repairing much more rewarding (you have that relief factor). After the first weekend of fun, what could I do for amusement? Porting over some of my favorite utilities would be fun. Both Red Hat and Craftworks take some of the fun out by already porting over the Apache Web Server and browsers, but I could still have some fun by moving other stuff over. So, the next question is, “What do you have to know to port code to Alpha Linux?”

### Porting Hints

X-Windows applications that use IMakefiles basically configure themselves out of the box. GNU software that uses autoconf/configure to figure out what system it is running on tends to get confused. The machine configuration string that it synthesizes looks like **alpha-unknown-linuxaout**. This is confusing because it is an alpha that is not running DEC Unix, nor is it Linux running on an

Intel system. What to do? Well, I usually put in the following code segment into configure:

```
  alpha-dec-osf3* )
      machine=alpha opsys=decosf3-1
    ;;
    ## We're Alpha Linux
    alpha-*-linux* )
      machine=alpha_linux opsys=linux_axp
   ;;

    ## Altos 3068
    m68*-altos-sysv* )
      machine=altos opsys=usg5-2
    ;;
```

But that means I have to write a **./src/m/alpha_linux.h** (which I would make by blending **alpha.h**, and removing anything cd DEC Unix specific), and **./src/s/linux_axp.h** (which would be made from **linux.h**, minus instructions on how to make sharable libraries). None of that is too difficult. Later releases of most software will come with pre-built configuration files as autoconf gets updated, and developers begin to use the new version.

The other issue you get involved with is the fact that several programs publicly available assume 32-bit addresses and 32-bit ints. Linux for the Alpha is a 64-bit operating system, with 64-bit addresses. Frequently, this provides harmless warnings about adding 32-bit offsets to 64-bit pointers.

Then there are the programs that attempt to override the definition of operating system calls. System calls that have been standardized to take parameters like **size_t** (but is being redefined for unsigned int) will cause complaints from the compiler.

The really insidious things, though, are those programs that do bitwise manipulations without regard to portability. Generally, I've learned to become suspicious of any program that isn't packaged with autoconf/IMakefile, which runs only on one platform (e.g., it runs on Linux; you tell us if it runs on Solaris, BSD, HPUX, etc).

The Future Looks Bright

Many packages compile out of the box. With ELF support comes the ability to port the Java JDK over. Sun, HP, and other notables are releasing their 64-bit processors to the marketplace. And while everyone argues what the 64-bit standard for Unix will be, we will already have been there, and have moved on to more interesting projects.

**Bryan W. Headley** (bheadley@interaccess.com) has been working with Unix since 1978 except for an interruption by that interloper, MS-DOS. A Unix

applications developer by day, he becomes a Linux hacker by night. There isn't a compiler or kernel that he doesn't find worth playing with.

Archive Index Issue Table of Contents

Advanced search

# DEC AXP Review

**Bryan Phillippe**

Issue #30, October 1996

Faster than a speeding bullet, able to leap tall buildings... it's Digital's AXP (aka Alpha) Computer!

The Alpha computer I played with for this review was loaned to SSC by DCG Computers, Inc. of Londenderry, NH. DCG can be reached via e-mail at sjg@dcginc.com or by a visit to their web site at www.dcginc.com. DCG's goal is to produce affordable computers based on Digital's Alpha Technology. The loaned AXP was a model EV5-333Mhz with 64 MB of RAM, a 1 GB hard drive and a quad speed CD ROM. DCG markets it for about $7,100.

My first hands on experience with the AXP was trying to fix it—always a good way to start. The AXP EB164 had arrived, and had been mysteriously “broken” during a second boot attempt. Evidently, the first boot had worked fine, but someone in the office had changed a BIOS setting, rebooted and “Kaboom...Alpha bits.” Now, although everything appeared normal, the system would not even display the BIOS information. The hardware was functional, and the cards, SIMMs and cables were properly seated.

The system I wanted to install was Craftworks Linux, and since no one would 'fess up to changing the BIOS setting, much less to what it had been changed, I called Craftwork's Tech Support. The Tech Support people were friendly and helpful, and even gave me the answer to the problem—always a plus—the BIOS of the AXP has hard coded operating-system-dependent boot logic, i.e., the AXP only knows how to boot DEC Unix and Windows NT. The “someone” had set the BIOS switch to DEC Unix from NT, a perfectly reasonable thing to do considering the options. Wrong! For a Linux installation, the switch must start out set to NT, then you create and format a super tiny MS-DOS FAT partition (as partition 1), make it bootable and install the AXP equivalent of [cw]loadlin[ecw] in it. This hack will you get you booted okay, but it would be nice to have a Linux option to begin with.

After the initial boot problems were fixed, so was the AXP, and we started the installation of Craftworks 2.1 version of Linux. Due to the special AXP BIOS feature and the fact that MS-DOS was involved in some way, the installation was something other than “smooth and refreshing”. One unsuccessful installation was caused by veering from the suggested partition size values. The second installation went fine, as I used all the suggested sizes and followed all the steps as given.

After completing the installation, I was able to boot the AXP from its own drive and log in seconds later. It was incredibly fast—RISC fast—333Mhz fast—faster than a speeding bullet fast (well, almost)! Using the AXP was a lot of fun—no waiting on your prompt to come back, it was always there. The AXP was so fast (see below) and performed so well, it was like a dream, then our next stumbling block appeared. Following a kernel build of 4-5 minutes, I typed [cw]make clean.[ecw]. Shortly after make began execution, the script died with a segmentation fault. Following this failure, many of the regular, previously working utilities (e.g., ps, finger, telnet, inetd, init) began failing. The only fix I found was to reboot. Everything would run fine for hours, even through concurrent compiles of GhostScript and the kernel, but as soon as I would run the **make clean** script, everything would quit working. Through various experiments, I found that any embedded **rm** command would cause a segmentation fault, after which the system was totally unstable. I further found that this problem was a bug in the version of Linux that I was using. The problem went away with an upgrade of the kernel to a later version (1.3.89).

Our systems administrator, Liem Bahneman, ran a lot of benchmarks, including dhrystone 1.1, iozone (HDD performance), bonnie (HDD performance), and Byte UNIX benchmarks. Unfortunately, none yielded accurate results due to a known flaw with math in the C library (libm) as ported to Linux/AXP at that time. Another “benchmark” he ran was a render of a povray animation (i.e., ray tracing). To quote Liem, “From my estimates, the same 72-frame, 320X240 animation render that took 13 hours on a Sparc20 would have taken approximately 4 hours on the AXP as tested. This is an estimate, because the render could not complete due to math faults.”

I then installed Red Hat's version 3.0.3 of Linux finding it quite similar to Craftworks. Again, installation required the use of MILO (similar in functionality to LILO) and a DOS FAT partition. Following the guidelines carefully, I had it up and running quickly with no problems. Well after all, I was now an expert, having done it once before. Red Hat for the AXP looks and feels like Red Hat for the x86 except, of course, it's much faster. Red Hat's version of Linux did not have the embedded **rm** bug described above.

With Linux becoming more robust every day, the AXP running Linux will be a solid network daemon/file server. Especially since math-intensive applications are no longer flawed by a libm bug (this bug has now been fixed), and all “speed-matters” applications are math-intensive. If you are shopping for the high-end performance offered by the AXP, this is the machine for you.

**Bryan Phillippe** (bryan@terran.org) is a 21-year-old Linux enthusiast who also enjoys the company of his fiancee, rollerblading and street-style snowboarding.

Archive Index Issue Table of Contents

Advanced search

# Letters to the Editor

**Various**

Issue #30, October 1996

Readers sound off.

## On with the Masquerade

I read with interest Chris Kostick's article in *Linux Journal* July 1996 (Issue 27). I was a bit disappointed that he concentrated on 1.2.x kernels and didn't mention the advantages of using a 1.3.85+ or pre2.x kernel, and that he didn't explain the use of ipfwadm. For instance, although many ICMP packet programs do not work, with recent kernels ICMP support has been added for some programs such as traceroute (ping still does not, and probably never will work).

I was also bothered by his insistence that you couldn't pass remote X clients to a masqueraded server. I was sure you could, because I remembered doing it! Using ssh (a secure rlogin variant available at http://www.cs.hut.fi/ssh/) to connect to a remote host from the masqueraded server, your X display environment variable is automatically set, and "X11 connection forwarding provides secure X11 sessions !"

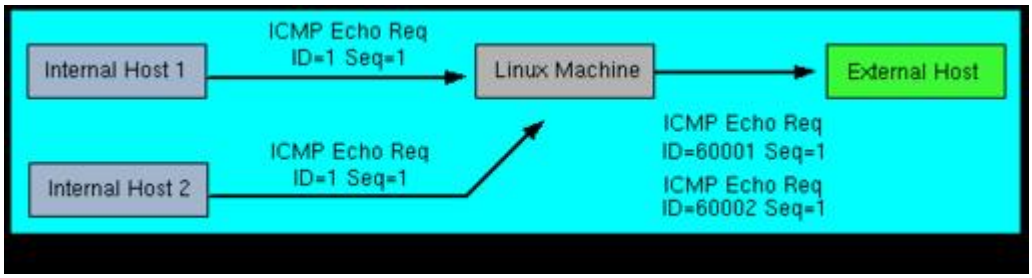
So you can pass X clients through a masquerading machine if you initiate a connection from the masqueraded machine using ssh.

—Daniel Morrison daniel@hartco.ca

## The Author Responds

At the time I wrote the article, 1.3.57 was the latest "stable" release. The 60's were available but most of them were to be avoided. I was using 1.3.56 so that's what I based the article on, pointing out that 1.2.x kernels were compatible.

Actually, ping could work. ICMP echo request/echo reply messages can be masqueraded for. The key is the identifier field in the ICMP message. The problem is matching replies that come back from the external networks to the originator. The masquerading would function as shown in Figure 1.




The masquerading machine will have to maintain a cache that will map the originating IP_Addr+ID --> ID numbers of the requests originating from the Masq host. The ID numbers start at a value that should not conflict with 'real' ICMP echos from the Masq host. I chose 60000 to keep in concert with the masquerading port numbers. The ICMP code within the kernel would also have to be modified so as not to conflict with internal ID numbers never exceeding 59999.

Matching the replies is a matter of checking the cache for the return mapping, and sending the ICMP echo reply to the original host.

There are other protocol details not discussed here, and I haven't looked at all of the nuances of the protocol so I leave the theory of whether it would really work up for discussion.

I didn't consider passing X (or any other protocol) encapsulated the same as sending the real thing. Therefore because the X client has to identify the address of the server in the DISPLAY variable or -display argument, and the server's address is hidden, it won't work.

—Chris Kostick ckostick@csc.com

### Virus Cleaners May Like DOS

Thanks again for another great issue of the *Linux Journal*. On the day it arrives all activity in the house (on my part anyway) stops until I have read it from front to back.

I would like to make a comment on Michael Johnson's article *Serving Two Masters*. The “Recovery Recipe” that Michael presents was something that I had to discover the hard way myself a month or two ago. The situation had nothing to do with Win95, although it did involve a dual boot PC. This PC runs both Linux and DOS/Win3.1 (thank you LILO), and in one of the DOS sessions I

managed to acquire a virus that infected the MBR. No problem says I—I'll just use McAfee Virus Clean, and all will be back to normal. Virus Clean worked fine, but all of a sudden LILO was gone. It appears that (at least with McAfee) the process of cleaning a boot sector virus just restores the DOS flavour MBR. I was not brave enough to re-infect my PC with the same virus to see if other virus cleaning programs act the same way, but I would suspect that they do (as most do not take periodic copies of the MBR to restore from).

I found the second edition of Æleen Frisch's book *Essential System Administration* (published by O'Reilly) absolutely invaluable in this effort. If you are a Linux user—run, don't walk and get this book!! Make sure that it is the second edition though as the first edition does not really deal with Linux.

—Kris Boyle kboyle@synapse.net

Archive Index Issue Table of Contents

Advanced search

# The Politics of Freedom

**Phil Hughes**

Issue #30, October 1996

In much the same way that the various political parties in the United States want to define what democracy really is, software politicians want to define what free software really is.

I like say it's back, but the reality is that it never went away. In much the same way that the various political parties in the United States want to define what democracy really is, software politicians want to define what free software really is.

While the majority of the users of free software (whether it be GPLed, public domain, BSD licensed or any of the other free classes of software) are happy to use it and appreciate that it exists, there is a minority who have their own political agenda. And the one at the top of the minority list is Richard M. Stallman (commonly known as RMS), creator of the Free Software Foundation.

Most Linux users are aware of the GNU project of the Free Software Foundation the people who have made a huge contribution to the Linux effort through the creation of programs like Emacs and gcc. But not many people know that the GNU project was supposed to turn out a complete Unix-like operating system, including a kernel called The Hurd.

RMS approached *Linux Journal* almost two years ago and told us that we should refer to what we call Linux as GNU/Linux. While we all recognize the contribution that the GNU project has made, we declined to make this change, as it is our job to report what is happening, not to create news (something that many other magazines along with newspapers and TV news programs could learn from).

Stallman's latest idea is to rename Linux as Lignux. (I would have included particulars from the opinion RMS wrote, but it is under a copyright that allows verbatim copying only, and it is longer than the space allotted here. The gist of

his stated opinion is that the GNU project has been working for 12 years to make something like what Linux is today, that Linux is based on GNU, and that the GNU project was built from other free software including X Windows, TeX and BSD network utilities. He then concludes that these components together make up what is called the GNU system.

Using this same logic we could say that we have combined the GNU system, the Linux kernel and other free software to produce what is called the Linux system. If fact, we do say it.

### What Went Wrong?

Perhaps RMS is frustrated because Linus got the glory for what RMS wanted to do. Linus managed to get more people working together for free to produce a commercial-grade finished product. While Linux didn't start out to be put under GPL license teRMS, Linus decided that was the right thing to do. Rms should see this as a serious conversion—it's like Linus found religion. Linux isn't a threat or a competitor; it is RMS's biggest success.

### What's All the Fuss About?

Or, put another way, why am I taking up all this space to discuss this matter? Because a split between FSF-supporters and Linux-supporters just doesn't benefit anyone in the free software community. It doesn't benefit any consumers of software—free or otherwise. In fact, it only benefits companies like Microsoft.

I have been a supporter of the Free Software Foundation for years: SSC continues to sell FSF books (none of which have ever been profitable for us, but have helped the FSF), and I have little disagreement with what RMS has to say. However, I do have a problem when he feels that everyone has to believe exactly the same things he does. I want Bill Gates to yield to the pressure of a successful free software movement and make his software freely available, rather than let him watch the infighting in the free software community over which type of free software is best.

I see two driving forces that have made Linux a success: it's good, and it exemplifies the right attitude. While the BSD crowd has been busy with infighting, and the Free Software Foundation has been trying to define what 'free' really is, the Linux community has been writing code and building a complete and successful package.

This attitude is what has made commercial vendors see Linux as a viable platform for their products. We've seen you can't go wrong if you buy IBM and then you can't go wrong if you buy Microsoft. Neither of these may have been

the best answer, but both were safe answers. Linux is becoming successful in commercial markets, because once again, it offers a safe answer.

While I don't have a formula that will make all this go away (so we can get back to development on the best operating system around that just happens to contain software from various sources including the Free Software Foundation), I would like to offer one final opinion; this one is from Darin Johnson's Usenet posting, where he says, heck, you can do whatever you want with my posting. I think it offers another way to look at what is happening:

Finally, to quote someone I think we all know: “Umm, this discussion has gone on quite long enough, thank you very much. It doesn't really matter what people call Linux, as long as credit is given where credit is due (on both sides). Personally, I'll very much continue to call it *Linux*.”

Relax, Linus—so will we.

Archive Index Issue Table of Contents

Advanced search

# New Column: Linux Means Business

**Phil Hughes**

Issue #30, October 1996

Introducing 'Linux Means Business' column.

It has been called to my attention that we didn't introduce the *Linux Means Business* column that first appeared in *Linux Journal* issue number 26, June 1996. So without further delay, here is a belated introduction.

Anyone who has used Linux needs little convincing that it is a real software product: it's a complete system; it's reliable; it's available from multiple sources; and it's documented.

However, the Linux operating system has a problem—it's free. That one fact means some people will not take it seriously. When reading the Usenet newsgroups, I continue to see people concerned with questions like “Is Linux good enough to do my task?” or “How can I convince my boss that Linux is real?”

While our (occasional) *Linux in the Real World* column has touched on many places where Linux offered a solution, it has tended to be more of an example of how one technical person managed to use Linux as a base to do something fairly unique for them. These columns contain good information. With LMB, however, we want to address different issues—what might be considered ordinary business problems.

For example, in this issue, LMB covers how a large corporation has used Linux to reduce the complexities of their electronic mail handling system. This solution did not require any add-on software, only the configuring of a Linux system to do the task at hand. This sort of article can offer a solution to two different issues: the job addressed in the article as well as the job of convincing management that Linux is a viable software system.

When we came up with the idea for the column, Gena Shurtleff sent out a query to Usenet looking for articles. Response has been overwhelming; therefore, we are planning a future issue of *Linux Journal* that will focus on articles in which Linux has been installed as a solution to a particular problem. This issue will include both the *Linux Means Business* and the *Linux in the Real World* articles. If you have a story to contribute, send it to us via e-mail at info@linuxjournal.com.

Archive Index Issue Table of Contents

Advanced search

# Let's Talk About the Competition

**Phil Hughes**

Issue #30, October 1996

It isn't about Linux. Or is it?

I had planned to write this column about what has been happening with Linux since 2.0 was released. I was going to cover the updates, as of today, to Linux 2.0.7. But, I forgot—updates in a stable chain are uninteresting, as they are just bug fixes.

Fortunately, three pieces of news caught my eye, and I now had topics that are both current and of interest. But, it isn't about Linux. Or is it?

## Where is UnixWare Headed?

First, for newcomers to the Unix racket, UnixWare is a product of Unix Systems Labs (USL). AT&T sold USL to Novell and last year Novell sold USL to SCO. So, UnixWare is what remains of what many considered “Real Unix” from AT&T.

In the ITbits newsletter published by Implements, Inc., Norton Greenfeld comments that the UnixWare Technology Group (UTG) is being dissolved. SCO (the guys who now own UnixWare) will form an internal group to take its place. What's wrong with this? UTG was reasonably independent, and its decisions had to do with the entire Unix industry. Independence is no longer the case, as SCO has just put themselves in the position of being on all sides of any decisions. For later reference, note that a little company in Redmond named Microsoft owns a reasonably-sized chunk of SCO.

## On to Windows NT

I just received a release (called an alert) to journalists and analysts from Tim O'Reilly of O'Reilly & Associates titled *NT Workstations 4.0: Bad News for Web Servers*. The title immediately got my attention, but it turns out that rather than a warning that NT Workstations 4.0 will destroy your web site, it was a call for

political action. The gist of the alert is that this new version of NT is designed to limit performance so that you can't use it as a web server. This means you will have to purchase NT Server (for $999 instead of only $290 for NT Workstations) if you want to run a web server. Tim goes on to quote web site developer Bob Denny: “When I first started developing web servers in 1994, nearly all web serving was done on the Unix platform. Considering that companies such as O'Reilly & Associates, Netscape and half a dozen more, pushed hard in the fight to legitimize NT vs. Unix as a web server platform over the last 18 months, Microsoft's actions are pretty extreme.”

### What Does This Have to Do with Linux?

In case you haven't caught on to where I am going with this—it isn't where Tim was going. He thinks this is bad; I think this is an opportunity. A few days ago I was talking to a vendor of Alpha systems running Linux. He told me that even Digital was surprised at the number of systems he was selling. Many of these systems are for web servers.

Linux people, now is the time to strike. Linux is a great operating system for web servers. Our own web server is a 486DX4/100-based Linux system. Our site has grown in popularity to around 80,000 hits a day, and the server continues to perform flawlessly. By the time you read this we should have the secure version of the Apache server running on it.

For a higher-powered web server an Alpha-based Linux system offers more performance at a lower cost than NT. In other words, everything that NT can do can also be done by this system.

We have a chance to show the big guys that we know what we are doing. Selling a Unix-like platform to the Internet community isn't hard. After all, the Internet grew up on such platforms.

When MS-DOS was the $100 answer against the $1000 Xenix answer, people were picking MS-DOS. Note that I am ignoring capabilities and performance. Capabilities and performance seem to seldom enter into mass marketing efforts anyway. (Remember Beta VCRs offered superior performance at the same cost.)

Today the game is different. While Microsoft is trying to get $1000 out of your pocket, Linux offers a much less expensive alternative that generally performs as well or better.

## Remember DR DOS?

DR DOS is/was the MS-DOS-like system that was developed by Digital Research. It was bought by Novell, had a significant following for a while and then faded away over the last two years.

Well, Caldera just acquired DR DOS from Novell, where “just” means July 24. What does this have to do with Linux? A lot. Aside from the fact that Caldera is a Linux company, part of the DR DOS package was a lawsuit (filed on July 23) against Microsoft.

Microsoft is accused of “illegal conduct ... calculated and intended to prevent and destroy competition in the computer software industry.” Of particular interest to the Linux community are the deals that Microsoft has cut to get OEMs to include MS-DOS with the computer. While I doubt anyone reading this is very excited about having DR DOS as an alternative to MS-DOS on their new system, this action could open the way for other operating systems (for example, Linux) being more widely available. (For more information, check out www.caldera.com/news/pr001.html.)

## A Call to Action

What do I want you to do? Sell Linux. By that I mean talk to your boss who is considering setting up a web server. Talk to your ISP. Point them to some examples. (www.ssc.com is our web server, but don't stop there. Our ISP, running Linux systems can be found at www.aa.net/. And there are certainly more.)

There may be a lot of copies of software by Microsoft out there, but Linux is a significant influence on the Internet and now is a good time for us to make sure it stays that way.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL



# Network Buffers and Memory Management

**Alan Cox**

Issue #30, October 1996

Writing a network device driver for Linux is fundamentally simple—most of the complexity (other than talking to the hardware) involves managing network packets in memory.

The Linux operating system implements the industry-standard Berkeley socket API, which has its origins in the BSD Unix developments (4.2/4.3/4.4 BSD). In this article, we will look at the way the memory management and buffering is implemented for network layers and network device drivers under the existing Linux kernel, as well as explain how and why some things have changed over time.

## Core Concepts

The networking layer is fairly object-oriented in its design, as indeed is much of the Linux kernel. The core structure of the networking code goes back to the initial networking and socket implementations by Ross Biro and Orest Zborowski respectively. The key objects are:

- **Device or Interface:** A network interface is programming code for sending and receiving data packets. Usually an interface is used for a physical device like an Ethernet card; however, some devices are software only, e.g., the loopback device used for sending data to yourself.
- **Protocol:** Each protocol is effectively a different networking language. Some protocols exist purely because vendors chose to use proprietary networking schemes, while others are designed for special purposes. Within the Linux kernel each protocol is a separate module of code which provides services to the socket layer.
- **Socket:** A socket is a connection in the networking that provides Unix file I/O and exists as a file descriptor to the user program. In the kernel each socket is a pair of structures that represent the high level socket interface and the low level protocol interface.

- **sk_buff:** All the buffers used by the networking layers are **sk_buff**s. The control for these buffers is provided by core low-level library routines that are available to all of the networking system. **sk_buff**s provide the general buffering and flow control facilities needed by network protocols.

### Implementation of sk_buffs

The primary goal of the **sk_buff** routines is to provide a consistent and efficient buffer-handling method for all of the network layers, and by being consistent to make it possible to provide higher level **sk_buff** and socket handling facilities to all of the protocols.

A **sk_buff** is a control structure with a block of memory attached. Two primary sets of functions are provided in the **sk_buff** library. The first set consists of routines to manipulate doubly linked lists of **sk_buffs**; the second of functions for controlling the attached memory. The buffers are held on linked lists optimised for the common network operations of append to end and remove from start. As so much of the networking functionality occurs during interrupts these routines are written to use atomic memory. The small extra overhead that results is well worth the pain it saves in bug hunting.

We use the list operations to manage groups of packets as they arrive from the network, and as we send them to the physical interfaces. We use the memory manipulation routines for handling the contents of packets in a standardised and efficient manner.

At its most basic level, a list of buffers is managed using functions like this:

```
void append_frame(char *buf, int len)
{
  struct sk_buff *skb=alloc_skb(len, GFP_ATOMIC);
  if(skb==NULL)
    my_dropped++;
  else
  {
    skb_put(skb,len);
    memcpy(skb->data,data,len);
    skb_append(&my_list, skb);
  }
}
void process_queue(void)
{
  struct sk_buff *skb;
  while((skb=skb_dequeue(&my_list))!=NULL)
  {
    process_data(skb);
    kfree_skb(skb, FREE_READ);
  }
}
```

These two fairly simplistic pieces of code actually demonstrate the receive packet mechanism quite accurately. The **append_frame()** function is similar to the code called from an interrupt by a device driver receiving a packet, and **process_frame()** is similar to the code called to feed data into the protocols. If

you look in net/core/dev.c at **netif_rx()** and **net_bh()**, you will see that they manage buffers similarly. They are far more complex, as they have to feed packets to the right protocol and manage flow control, but the basic operations are the same. This is just as true if you look at buffers going from the protocol code to a user application.

The example also shows the use of one of the data control functions, **skb_put()**. Here it is used to reserve space in the buffer for the data we wish to pass down.

Let's look at **append_frame()**. The **alloc_skb()** function obtains a buffer of **len** bytes (Figure 1), which consists of:

- 0 bytes of room at the head of the buffer
- 0 bytes of data, and
- **len** bytes of room at the end of the data.

The **skb_put()** function (Figure 4) grows the **data** area upwards in memory through the free space at the buffer end, and thus reserves space for the **memcpy()**. Many network operations that send data packets add space to the start of the frame each time a send is executed, so that headers can be added to the packets. For this reason, the **skb_push()** function (Figure 5) is provided so that the start of the data frame can be moved down through memory, if enough space has been reserved to leave room for completing this operation.

Figure 1 “After alloc_skb”

Figure 2 “After skb_reserve”

Figure 3 “An sk_buff Containing Data”

Figure 4 “After skb_put has been Called on the Buffer”

Figure 5 “After an skb_push has Occurred on the Previous Buffer”

Immediately after a buffer has been allocated, all the available room is at the end. Another function, **skb_reserve()** (Figure 2), can be called before data is added. This function allows you to specify that some of the space should be at the beginning of the buffer. Thus, many sending routines start with code that looks like:

```
skb=alloc_skb(len+headspace, GFP_KERNEL);
skb_reserve(skb, headspace);
skb_put(skb,len);
memcpy_fromfs(skb->data,data,len);
pass_to_m_protocol(skb);
```

In systems such as BSD Unix, you don't need to know in advance how much space you will need, as it uses chains of small buffers (mbufs) for its network buffers. Linux chooses to use linear buffers and save space in advance (often wasting a few bytes to allow for the worst case), because linear buffers make many other operations much faster.

Linux provides the following functions for manipulating lists:

- **skb_dequeue()** takes the first buffer from a list. If the list is empty, a **NULL** pointer is returned. This function is used to pull buffers off queues. The buffers are added with the routines **skb_queue_head()** and **skb_queue_tail()**.
- **skb_queue_head()** places a buffer at the start of a list. As with all the list operations, it is atomic.
- **skb_queue_tail()** places a buffer at the end of a list and is the most commonly used function. Almost all the queues are handled with one set of routines queuing data with this function and another set removing items from the same queues with **skb_dequeue()**.
- **skb_unlink()** removes a buffer from whatever list contains it. The buffer is not freed, merely removed from the list. To make some operations easier, you need not know what list the buffer is in, and you can always call **skb_unlink()** for a buffer which is not in any list. This function enables network code to pull a buffer out of use even when the network protocol has no idea who is currently using the buffer. A separate locking mechanism is provided, so that a buffer currently in use by a device driver can not be removed.
- Some more complex protocols, like TCP, keep frames in order, and re-order their input as data is received. Two functions, **skb_insert()** and **skb_append()**, exist to allow users to place **sk_buff**s before or after a specific buffer in a list.
- **alloc_skb()** creates a new **sk_buff** and initializes it. The returned buffer is ready to use but assumes you will fill in a few fields to indicate how the buffer should be freed. Normally this is done by **skb->free=1**. A buffer can be flagged as not freeable by **kfree_skb()** (see below).
- **kfree_skb()** releases a buffer, and if **skb->sk** is set, it lowers the memory use counts of the socket (**sk**). It is up to the socket and protocol-level routines to increment these counts and to avoid freeing a socket with outstanding buffers. The memory counts are very important, as the kernel networking layers need to know how much memory is tied up by each connection in order to prevent remote machines or local processes from using too much memory.
- **skb_clone()** makes a copy of a **sk_buff**, but does not copy the data area, which must be considered read only.

- Sometimes a copy of the data is needed for editing, and **skb_copy()** provides the same facilities as **skb_clone**, but also copies the data (and thus has a much higher overhead).

Figure 6 Flow of Packets

### Higher Level Support Routines

The semantics of allocating and queuing buffers for sockets also involve flow control rules and for sending a whole list of interactions with signals and optional settings such as non blocking. Two routines are designed to make this easy for most protocols.

The **sock_queue_rcv_skb()** function is used to handle incoming data flow control and is normally used in the form:

```
sk=my_find_socket(whatever);
if(sock_queue_rcv_skb(sk,skb)==-1)
{
    myproto_stats.dropped++;
    kfree_skb(skb,FREE_READ);
    return;
}
```

This function uses the socket read queue counters to prevent vast amounts of data from being queued to a socket. After a limit is hit, data is discarded. It is up to the application to read fast enough, or as in TCP, for the protocol to do flow control over the network. TCP actually tells the sending machine to shut up when it can no longer queue data.

On the sending side, **sock_alloc_send_skb()** handles signal handling, the non-blocking flag and all the semantics of blocking until there is space in the send queue, so that you cannot tie up all of memory with data queued for a slow interface. Many protocol send routines have this function doing almost all the work:

```
skb=sock_alloc_send_skb(sk,....)
if(skb==NULL)
    return -err;
skb->sk=sk;
skb_reserve(skb, headroom);
skb_put(skb,len);
memcpy(skb->data, data, len);
protocol_do_something(skb);
```

Most of this we have met before. The very important line is **skb->sk=sk**. The **sock_alloc_send_skb()** has charged the memory for the buffer to the socket. By setting **skb->sk**, we tell the kernel that whoever does a **kfree_skb()** on the buffer should credit the memory for the buffer to the socket. Thus, when a device has sent a buffer and freed it, the user is able to send more.

## Network Devices

All Linux network devices follow the same interface, but many functions available in that interface are not needed for all devices. An object-oriented mentality is used, and each device is an object with a series of methods that are filled into a structure. Each method is called with the device itself as the first argument, in order to get around the lack of the C++ concept of **this** within the C language.

The file drivers/net/skeleton.c contains the skeleton of a network device driver. View or print a copy from a recent kernel and follow along throughout the rest of the article.

## Basic Structure

Figure 7 Structure of a Linux Network Device

Each network device deals entirely in the transmission of network buffers from the protocols to the physical media, and in receiving and decoding the responses the hardware generates. Incoming frames are turned into network buffers, identified by protocol and delivered to **netif_rx()**. This function then passes the frames off to the protocol layer for further processing.

Each device provides a set of additional methods for the handling of stopping, starting, control and physical encapsulation of packets. All of the control information is collected together in the device structures that are used to manage each device.

## Naming

All Linux network devices have a unique name that is not in any way related to the file system names devices may have. Indeed, network devices do not normally have a file system representation, although you can create a device which is tied to the device drivers. Traditionally the name indicates only the type of a device rather than its maker. Multiple devices of the same type are numbered upwards from 0; thus, Ethernet devices are known as “eth0”, “eth1”, “eth3” etc. The naming scheme is important as it allows users to write programs or system configuration in terms of “an Ethernet card” rather than worrying about the manufacturer of the board and forcing reconfiguration if a board is changed.

The following names are currently used for generic devices:

- eth*n* Ethernet controllers, both 10 and 100Mbit/second
- tr*n* Token ring devices

- sl*n* SLIP devices and AX.25 KISS mode
- ppp*n* PPP devices both asynchronous and synchronous
- plip*n* PLIP units; the number matches the printer port
- tunl*n* IPIP encapsulated tunnels
- nr*n* NetROM virtual devices
- isdn*n* ISDN interfaces handled by isdn4linux (*)
- dummy*n* Null devices
- lo The loopback device

(*) At least one ISDN interface is an Ethernet impersonator—the Sonix PC/Volante driver behaves in all aspects as if it was Ethernet rather than ISDN; therefore, it uses an “eth” device name. If possible, a new device should pick a name that reflects existing practice. When you are adding a whole new physical layer type, you should look for other people working on such a project and use a common naming scheme.

Certain physical layers present multiple logical interfaces over one media. Both ATM and Frame Relay have this property, as does multi-drop KISS in the amateur radio environment. Under such circumstances, a driver needs to exist for each active channel. The Linux networking code is structured in such a way as to make this manageable without excessive additional code. Also, the name registration scheme allows you to create and remove interfaces almost at will as channels come into and out of existence. The proposed convention for such names is still under some discussion, as the simple scheme of “sl0a”, “sl0b”, “sl0c” works for basic devices like multidrop KISS, but does not cope with multiple frame relay connections where a virtual channel can be moved across physical boards.

### Registering a Device

Each device is created by filling in a **struct device** object and passing it to the **register_netdev(struct device *)** call. This links your device structure into the kernel network device tables. As the structure you pass in is used by the kernel, you must not free this until you have unloaded the device with **void unregister_netdev(struct device *)** calls. These calls are normally done at boot time or at module load/unload.

The kernel will not object if you create multiple devices with the same name, it will break. Therefore, if your driver is a loadable module you should use the **struct device *dev_get(const char *name)** call to ensure the name is not already in use. If it is in use, you should pick another name or your new driver will fail. If you discover a clash, you must not use **unregister_netdev()** to unregister the other device using the name!

A typical code sequence for registration is:

```
int register_my_device(void)
{
  int i=0;
  for(i=0;i<100;i++)
  {
    sprintf(mydevice.name,"mydev%d",i);
    if(dev_get(mydevice.name)==NULL)
    {
      if(register_netdev(&mydevice)!=0)
        return -EIO;
      return 0;
    }
  }
  printk(
"100 mydevs loaded. Unable to load more.\n");
  return -ENFILE;
}
```

### The Device Structure

All the generic information and methods for each network device are kept in the device structure. To create a device you need to supply the structure with most of the data discussed below. This section covers how a device should be set up.

### Naming

First, the name field holds a string pointer to a device name in the formats discussed previously. The name field can also be **" "** (four spaces), in which case the kernel automatically assigns an eth*n* name to it. This special feature should not be used. After Linux 2.0, we intend to add a simple support function of the form **dev_make_name("eth")** for this purpose.

### Bus Interface Parameters

The next block of parameters is used to maintain the location of a device within the device address spaces of the architecture. The **irq** field holds the interrupt (IRQ) the device is using, and is normally set at boot time or by the initialization function. If an interrupt is not used, not currently known or not assigned, the value zero should be used. The interrupt can be set in a variety of fashions. The auto-irq facilities of the kernel can be used to probe for the device interrupt, or the interrupt can be set when loading the network module. Network drivers normally use a global int called **irq** for this so that users can load the module with **insmod mydevice irq=5** style commands. Finally, the IRQ field can be set dynamically using the **ifconfig** command, which causes a call to your device that will be discussed later on.

The **base_addr** field is the base I/O space address where the device resides. If the device uses no I/O locations or is running on a system without an I/O space concept, this field should be set to zero. When this address is user settable, it is

normally set by a global variable called **io**. The interface I/O address can also be set with **ifconfig**.

Two hardware-shared memory ranges are defined for things like ISA bus shared memory Ethernet cards. For current purposes, the **rmem_start** and **rmem_end** fields are obsolete and should be loaded with 0. The **mem_start** and **mem_end** addresses should be loaded with the start and end of the shared memory block used by this device. If no shared memory block is used, then the value 0 should be stored. Those devices that allow the user to to set the memory base use a global variable called **mem**, and then set the **mem_end** address appropriately themselves.

The **dma** variable holds the DMA channel in use by the device. Linux allows DMA (like interrupts) to be automatically probed. If no DMA channel is used or the DMA channel is not yet set, the value 0 is used. This option may have to change, since the latest PC boards allow ISA bus DMA channel 0 to be used by hardware boards and do not just tie it to memory refresh. If the user can set the DMA channel, the global variable **dma** is used.

It is important to realise that the physical information is provided for control and user viewing (as well as the driver's internal functions), and does not register these areas to prevent them being reused. Thus, the device driver must also allocate and register the I/O, DMA and interrupt lines it wishes to use, using the same kernel functions as any other device driver. [See the recent Kernel Korner articles on writing a character device driver in issues 23, 24, 25, 26 and 28, or visit the new Linux Kernel Hackers' Guide at www.redhat.com:8080/HyperNews/get/khg.html, for more information on the necessary functions—ED]

The **if_port** field holds the physical media type for multi-media devices such as combo Ethernet boards.

### Protocol Layer Variables

In order for the network protocol la

yers to perform in a sensible manner, the device has to provide a set of capability flags and variables that are also maintained in the device structure.

The **mtu** is the largest payload that can be sent over this interface, i.e., the largest packet size not including any bottom layer headers that the device itself will provide. This number is used by the protocol layers such as IP to select suitable packet sizes to send. There are minimums imposed by each protocol. A device is not usable for IPX without a 576 byte frame size or higher. IP needs at

least 72 bytes and does not perform sensibly below about 200 bytes. It is up to the protocol layers to decide whether to co-operate with your device.

The **family** is always set to **AF_INET** and indicates the protocol family the device is using. Linux allows a device to be using multiple protocol families at once, and maintains this information solely to look more like the standard BSD networking API.

The interface hardware **type** field is taken from a table of physical media types. The values used by the ARP protocol (see RFC1700) are used by those media that support ARP, and additional values are assigned for other physical layers. New values are added whenever necessary both to the kernel and to **net-tools**, the package containing programs like **ifconfig** that need to be able to decode this field. The fields defined as of Linux pre2.0.5 are:

```
From RFC1700:
ARPHRD_NETROM   NET/ROM™ devices
ARPHRD_ETHER         10 and 100Mbit/second Ethernet
ARPHRD_EETHER   Experimental Ethernet (not used)
ARPHRD_AX25          AX.25 level 2 interfaces
ARPHRD_PRONET        PROnet token ring (not used)
ARPHRD_CHAOS         ChaosNET (not used)
ARPHRD_IEE802        802.2 networks notably token ring
ARPHRD_ARCNET        ARCnet interfaces
ARPHRD_DLCI          Frame Relay DLCI
Defined by Linux:
ARPHRD_SLIP          Serial Line IP protocol
ARPHRD_CSLIP         SLIP with VJ header compression
ARPHRD_SLIP6         6bit encoded SLIP
ARPHRD_CSLIP6        6bit encoded header compressed SLIP
ARPHRD_ADAPT         SLIP interface in adaptive mode
ARPHRD_PPP      PPP interfaces (async and sync)
ARPHRD_TUNNEL   IPIP tunnels
ARPHRD_TUNNEL6  IPv6 over IP tunnels
ARPHRD_FRAD          Frame Relay Access Device
ARPHRD_SKIP          SKIP encryption tunnel
ARPHRD_LOOPBACK Loopback device
ARPHRD_LOCALTLK Localtalk apple networking device
ARPHRD_METRICOM Metricom Radio Network
```

Those interfaces marked unused are defined types but without any current support on the existing net-tools. The Linux kernel provides additional generic support routines for devices using Ethernet and token ring.

The **pa_addr** field is used to hold the IP address when the interface is up. Interfaces should start down with this variable clear. **pa_brdaddr** is used to hold the configured broadcast address, **pa_dstaddr** is the target of a point to point link, and **pa_mask** is the IP netmask of the interface. All of these can be initialized to zero. The **pa_alen** field holds the length of an address (in our case an IP address), and should be initialized to 4.

### Link Layer Variables

The **hard_header_len** is the number of bytes the device needs at the start of a network buffer passed to it. This value does not have to equal the number of

bytes of physical header that will be added, although this number is usually used. A device can use this value to provide itself with a scratch pad at the start of each buffer.

In the 1.2.x series kernels, the **skb->data** pointer will point to the buffer start, and you must avoid sending your scratch pad. This also means that for devices with variable length headers you need to allocate **max_size+1** bytes and keep a length byte at the start so that you know where the header actually begins (the header should be contiguous with the data). Linux 1.3.x makes life much simpler. It ensures that you have at least as much room as you requested, free at the start of the buffer. It is up to you to use **skb_push()** appropriately, as we discussed in the section on networking buffers.

The physical media addresses (if any) are maintained in **dev_addr** and **broadcast** respectively and are byte arrays. Addresses smaller than the size of the array are stored starting from the left. The **addr_len** field is used to hold the length of a hardware address. With many media there is no hardware address, and in this case, this field should be set to zero. For some other interfaces, the address must be set by a user program. The ifconfig tool permits the setting of an interface hardware address. In this case it need not be set initially, but the open code should take care not to allow a device to start transmitting before an address has been set.

### Flags

A set of flags is used to maintain the interface properties. Some of these are "compatibility" items and as such are not directly useful. The flags are:

- **IFF_UP** The interface is currently active. In Linux, the IFF_RUNNING and IFF_UP flags are basically handled as a pair, existing as two items for compatibility reasons. When an interface is not marked as IFF_UP, it can be removed. Unlike BSD, an interface that does not have IFF_UP set will never receive packets.
- **IFF_BROADCAST** The interface has broadcast capability. There will be a valid IP address for the interface stored in the device addresses.
- **IFF_DEBUG** Indicates debugging is desired. Not currently used.
- **IFF_LOOPBACK** The loopback interface (lo) is the only interface that has this flag set. Setting it on other interfaces is neither defined nor a very good idea.
- **IFF_POINTOPOINT** This interface is a point to point link (such as SLIP or PPP). There is no broadcast capability as such. The remote point to point address in the device structure is valid. Normally, a point to point link has no netmask or broadcast, but it can be enabled if needed.

- **IFF_NOTRAILERS** More of a prehistoric than an historic compatibility flag. Not used.
- **IFF_RUNNING** See **IFF_UP**
- **IFF_NOARP** The interface does not perform ARP queries. Such an interface must have either a static table of address conversions or no need to perform mappings. The NetROM interface is a good example of this. Here all entries are hand configured as the NetROM protocol cannot do ARP queries.
- **IFF_PROMISC** If it is possible, the interface will hear all of the packets on the network. This flag is typically used for network monitoring, although it can also be used for bridging. One or two interfaces like the AX.25 interfaces are always in promiscuous mode.
- **IFF_ALLMULTI** Receive all multicast packets. An interface, that cannot perform this operation but can receive all packets, will go into promiscuous mode when asked to perform this task.
- **IFF_MULTICAST** Indicates that the interface supports multicast IP traffic, which is not the same as supporting a physical multicast. AX.25 for example supports IP multicast using physical broadcast. Point to point protocols such as SLIP generally support IP multicast.

### The Packet Queue

Packets are queued for an interface by the kernel protocol code. Within each device, **buffs[]** is an array of packet queues for each kernel priority level. These are maintained entirely by the kernel code, but must be initialized by the device itself on boot up. The intialization code used is:

```
int ct=0;
while(ct<DEV_NUMBUFFS)
{
    skb_queue_head_init(&dev->buffs[ct]);
    ct++;
}
```

All other fields should be initialized to 0.

The device gets to select the queue length it needs by setting the field **dev->tx_queue_len** to the maximum number of frames the kernel should queue for the device. Typically this is around 100 for Ethernet and 10 for serial lines. A device can modify this dynamically, although its effect will lag the change slightly.

### Network Device Methods

Each network device has to provide a set of actual functions (methods) for the basic low level operations. It should also provide a set of support functions that

interface the protocol layer to the protocol requirements of the link layer it is providing.

### Setup

The **init** method is called when the device is initialized and registered with the system, in order to perform any low level verification and checking needed. It returns an error code if the device is not present, if areas cannot be registered or if it is otherwise unable to proceed. If the **init** method returns an error, the **register_netdev()** call returns the error code, and the device is not created.

### Frame Transmission

All devices must provide a transmit function. It is possible for a device to exist that cannot transmit. In this case, the device needs a transmit function that simply frees the buffer passed to it. The dummy device has exactly this functionality on transmit.

The **dev->hard_start_xmit()** function is called to provide the driver with its own device pointer and network buffer (a **sk_buff**) for transmitting. If your device is unable to accept the buffer, it should return 1 and set **dev->tbusy** to a non-zero value. This action will queue the buffer to be retried again later, although there is no guarantee that a retry will occur. If the protocol layer decides to free the buffer that the driver has rejected, then the buffer will not be offered back to the device. If the device knows the buffer cannot be transmitted in the near future, for example due to bad congestion, it can call **dev_kfree_skb()** to dump the buffer and return 0 indicating the buffer has been processed.

If there is space the buffer should be processed. The buffer handed down already contains all the headers, including link layer headers, necessary and need only be loaded into the hardware for transmission. In addition, the buffer is locked, which means that the device driver has absolute ownership of the buffer until it chooses to relinquish it. The contents of a **sk_buff** remain read-only, with the exception that you are guaranteed that the next/previous pointers are free, so that you can use the **sk_buff** list primitives to build internal chains of buffers.

When the buffer has been loaded into the hardware or, in the case of some DMA driven devices, when the hardware has indicated transmission is complete, the driver must release the buffer by calling **dev_kfree_skb(skb, FREE_WRITE)**. As soon as this call is made, the **sk_buff** in question may spontaneously disappear, and the device driver should not reference it again.

## Frame Headers

It is necessary for the high level protocols to append low level headers to each frame before queuing it for transmission. It is also clearly undesirable that the protocol know in advance how to append low level headers for all possible frame types. Thus, the protocol layer calls down to the device with a buffer that has at least **dev->hard_header_len** bytes free at the start of the buffer. It is then up to the network device to correctly call **skb_push()** and to put the header on the packet using the **dev->hard_header()** method. Devices with no link layer header, such as SLIP, may have this method specified as NULL.

The method is invoked by giving the buffer concerned, the device's pointers, its protocol identity, pointers to the source and destination hardware addresses and the length of the packet to be sent. As the routine can be called before the protocol layers are fully assembled, it is vital that the method use the length parameter, **not** the buffer length.

The source address can be NULL to mean “use the default address of this device”, and the destination can be NULL to mean “unknown”. If as a result of an unknown destination, the header can not be completed, the space should be allocated and any bytes that can be filled in should be filled in. The function must then return the negative of the bytes of header added. This facility is currently only used by IP when ARP processing must take place. If the header is completely built, the function must return the number of bytes of header added to the beginning of the buffer.

When a header cannot be completed the protocol layers will attempt to resolve the necessary address. When this situation occurs, the **dev->rebuild_header()** method is called with the address at which the header is located, the device in question, the destination IP address and the network buffer pointer. If the device is able to resolve the address by whatever means available (normally ARP), then it fills in the physical address and returns 1. If the header cannot be resolved, it returns 0 and the buffer will be retried the next time the protocol layer has reason to believe resolution will be possible.

## Reception

There is no receive method in a network device, because it is the device that invokes processing of such events. With a typical device, an interrupt notifies the handler that a completed packet is ready for reception. The device allocates a buffer of suitable size with **dev_alloc_skb()**, and places the bytes from the hardware into the buffer. Next, the device driver analyses the frame to decide the packet type. The driver sets **skb->dev** to the device that received the frame. It sets **skb->protocol** to the protocol the frame represents, so that the frame can be given to the correct protocol layer. The link layer header pointer is

stored in **skb->mac.raw**, and the link layer header removed with **skb_pull()** so that the protocols need not be aware of it. Finally, to keep the link and protocol isolated, the device driver must set **skb->pkt_type** to one of the following:

- **PACKET_BROADCAST** Link layer broadcast
- **PACKET_MULTICAST** Link layer multicast
- **PACKET_SELF** Frame to us
- **PACKET_OTHERHOST** Frame to another single host

This last type is normally reported as a result of an interface running in promiscuous mode.

Finally, the device driver invokes **netif_rx()** to pass the buffer up to the protocol layer. The buffer is queued for processing by the networking protocols after the interrupt handler returns. Deferring the processing in this fashion dramatically reduces the time interrupts are disabled and improves overall responsiveness. Once **netif_rx()** is called, the buffer ceases to be property of the device driver and can not be altered or referred to again.

Flow control on received packets is applied at two levels by the protocols. First, a maximum amount of data can be outstanding for **netif_rx()** to process. Second, each socket on the system has a queue which limits the amount of pending data. Thus, all flow control is applied by the protocol layers. On the transmit side a per device variable **dev->tx_queue_len** is used as a queue length limiter. The size of the queue is normally 100 frames, which is large enough that the queue will be kept well filled when sending a lot of data over fast links. On a slow link such as a slip link, the queue is normally set to about 10 frames, as sending even 10 frames is several seconds of queued data.

One piece of magic that is done for reception with most existing devices, and one that you should implement if possible, is to reserve the necessary bytes at the head of the buffer to land the IP header on a long word boundary. The existing Ethernet drivers thus do:

```
skb=dev_alloc_skb(length+2);
if(skb==NULL)
    return;
skb_reserve(skb,2);
/* then 14 bytes of ethernet hardware header */
```

to align IP headers on a 16 byte boundary, which is also the start of a cache line and helps give performance improvements. On the SPARC or DEC Alpha these improvements are very noticeable.

### Optional Functionality

Each device has the option of providing additional functions and facilities to the protocol layers. Not implementing these functions will cause a degradation in service available via the interface, but will not prevent operation. These operations split into two categories—configuration and activation/shutdown.

### Activation and Shutdown

When a device is activated (i.e., the flag **IFF_UP** is set), the **dev->open()** method is invoked if the device has provided one. This invocation permits the device to take any action such as enabling the interface that is needed when the interface is to be used. An error return from this function causes the device to stay down and causes the user's activation request to fail with an error returned by **dev->open()**

The **dev->open()** function can also be used with any device that is loaded as a module. Here it is necessary to prevent the device from being unloaded while it is open; thus, the **MOD_INC_USE_COUNT** macro must be used within the open method.

The **dev->close()** method is invoked when the device is ready to be configured down and should shut off the hardware in such a way as to minimise machine load (e.g., by disabling the interface or its ability to generate interrupts). It can also be used to allow a module device to be unloaded after it is down. The rest of the kernel is structured in such a way that when a device is closed, all references to it by pointer are removed, in order to ensure that the device can be safely unloaded from a running system. The close method is not permitted to fail.

### Configuration and Statistics

A set of functions provide the ability to query and to set operating parameters. The first and most basic of these is a **get_stats** routine which when called returns a struct **enet_statistics** block for the interface. This block allows user programs such as **ifconfig** to see the loading of the interface and any logged problem frames. Not providing this block means that no statistics will be available.

The **dev->set_mac_address()** function is called whenever a superuser process issues an ioctl of type **SIOCSIFHWADDR** to change the physical address of a device. For many devices this function is not meaningful and for others it is not supported. In these cases, set this function pointer to **NULL**. Some devices can only perform a physical address change if the interface is taken down. For these devices, check the **IFF_UP** flag, and if it is set, return **-EBUSY**.

The **dev->set_config()** function is called by the **SIOCSIFMAP** function when a user enters a command like **ifconfig eth0 irq 11**. It then passes an **ifmap** structure containing the desired I/O and other interface parameters. For most interfaces this function is not useful, and you can return NULL.

Finally, the **dev->do_ioctl()** call is invoked whenever an ioctl in the range **SIOCDEVPRIVATE** to **SIOCDEVPRIVATE+15** is used on your interface. All these ioctl calls take a struct **ifreq**, which is copied into kernel space before your handler is called and copied back at the end. For maximum flexibility any user can make these calls, and it is up to your code to check for superuser status when appropriate. For example, the PLIP driver uses these calls to set parallel port time out speeds in order to allow a user to tune the plip device for his machine.

### Multicasting

Certain physical media types, such as Ethernet, support multicast frames at the physical layer. A multicast frame is heard by a group of hosts (not necessarily all) on the network, rather than going from one host to another.

The capabilities of Ethernet cards are fairly variable. Most fall into one of three categories:

- No multicast filters. The card either receives all multicasts or none of them. Such cards can be a nuisance on a network with a lot of multicast traffic, such as group video conferences.
- Hash filters. A table is loaded onto the card giving a mask of entries for desired multicasts. This method filters out some of the unwanted multicasts but not all.
- Perfect filters. Most cards that support perfect filters combine this option with 1 or 2 above, because the perfect filter often has a length limit of 8 or 16 entries.

It is especially important that Ethernet interfaces are programmed to support multicasting. Several Ethernet protocols (notably Appletalk and IP multicast) rely on Ethernet multicasting. Fortunately, most of the work is done by the kernel for you (see net/core/dev_mcast.c).

The kernel support code maintains lists of physical addresses your interface should be allowing for multicast. The device driver may return frames matching more than the requested list of multicasts if it is not able to do perfect filtering.

Whenever the list of multicast addresses changes, the device drivers **dev->set_multicast_list()** function is invoked. The driver can then reload its physical tables. Typically this looks something like:

```
if(dev->flags&IFF_PROMISC)
    SetToHearAllPackets();
else if(dev->flags&IFF_ALLMULTI)
    SetToHearAllMulticasts();
else
{
    if(dev->mc_count<16)
    {
        LoadAddressList(dev->mc_list);
        SetToHearList();
    }
    else
        SetToHearAllMulticasts();
}
```

There are a small number of cards that can only do unicast or promiscuous mode. In this case the driver, when presented with a request for multicasts has to go promiscuous. If this is done, the driver must itself set the **IFF_PROMISC** flag in **dev->flags**.

In order to aid the driver writer, the multicast list is kept valid at all times. This simplifies many drivers, as a reset from an error condition in a driver often has to reload the multicast address lists.

### Ethernet Support Routines

Ethernet is probably the most common physical interface type that can be handled. The kernel provides a set of general purpose Ethernet support routines that such drivers can use.

**eth_header()** is the standard Ethernet handler for the **dev-hard_header** routine, and can be used in any Ethernet driver. Combined with **eth_rebuild_header()** for the rebuild routine it provides all the ARP lookup required to put Ethernet headers on IP packets.

The **eth_type_trans()** routine expects to be fed a raw Ethernet packet. It analyses the headers and sets **skb->pkt_type** and **skb->mac** itself as well as returning the suggested value for **skb->protocol**. This routine is normally called from the Ethernet driver receive interrupt handler to classify packets.

**eth_copy_and_sum()**, the final Ethernet support routine is internally quite complex, but offers significant performance improvements for memory mapped cards. It provides the support to copy and checksum data from the card into a **sk_buff** in a single pass. This single pass through memory almost eliminates the cost of checksum computation when used and improves IP throughput.

**Alan Cox** has been working on Linux since version 0.95, when he installed it in order to do further work on the AberMUD game. He now manages the Linux Networking, SMP, and Linux/8086 projects and hasn't done any work on AberMUD since November 1993.

# Using Sendmail as a Multi-Platform Mail Router

**Tom Lowery**

Issue #30, October 1996

See how one company uses a Linux system and sendmail to handle e-mail routing between incompatible systems.

Is e-mail the wave of the future? No way. It's a mission-critical, gotta-have application today. Survey those business cards you've been collecting from associates lately. Odds are most of them are sporting Internet e-mail addresses. In the April 1996 issue of *Microsoft Magazine*, Bill Gates said, "[E-mail] is probably the most mission-critical application for Microsoft in terms of running the company. If we had to pick one application that would keep running no matter what, E-mail would absolutely be it." People want fast, reliable, written communications with people in their own company as well as with others across the Net.

My job is to keep e-mail running smoothly for my company. A user recently complained to me because it took a whole 12 minutes for her e-mail message to be delivered. People have ever-growing expectations about what e-mail should do for them.

Providing e-mail in a homogeneous environment is simple. If a shop consists entirely of Linux workstations, e-mail is practically automatic. The same is true for all other major variants of Unix. No special gateways are required to connect to the Internet since the e-mail protocol spoken, SMTP, is the same across the board. SMTP stands for Simple Mail Transport Protocol and has been the de facto standard for Internet mail transmission for years. Exchanging mail is almost as simple with non-SMTP commercial packages, as long as everyone in the company uses the same one. The situation is more complex when several incompatible systems are in use. Getting mail to flow seamlessly between them and additionally the Internet can pose a lot of problems. In this article I will explain how Caliber Technology solved these problems quickly and inexpensively with Linux and sendmail.

Caliber System, through its operating units RPS, Viking Freight, Caliber Logistics, Roberts Express and Caliber Technology, is a value added provider of transportation, logistics, and information services. Caliber Technology provides integrated information services to customers, Caliber companies and other interested users.

### The Problem

The Caliber companies operate in vastly different computing environments: IBM mainframes, AS/400s, HP/UX midrange systems, Novell and NT LANs, Tandem minicomputers and more. The e-mail platforms in use include Microsoft Mail, Lotus cc:Mail, Tandem PS Mail, IBM OV/400, TAO/Emc2 and Microsoft Exchange. Unfortunately these mail systems are generally not compatible with each other. We have plans to migrate all the Caliber companies to compatible e-mail platforms, but the migration effort will take at least two years. The good news is gateways are available for all these systems. The function of a mail gateway is to translate messages from one format to another. Some gateways translate messages from one proprietary format to another: for example, from Lotus cc:Mail to Microsoft Mail and vice versa. Other gateways translate messages between a proprietary format and a standard format. An example here would be cc:Mail to and from SMTP. The two most common standard formats are SMTP and X.400. X.400 is the CCITT/ISO standard and is supported by many commercial networks. Since Caliber's goal was to have all its internal mail systems conversing with each other and with the Internet, SMTP was the obvious choice as the *common language*.

Here's where it gets complicated. Like many companies, Caliber Technology uses non-obvious login names for its mainframe and LAN systems. In a typical Unix environment, my login name might be **tlowery**.

But in Caliber's multi-platform environment my login name is, say, **xyz123**. This was a legacy decision to provide better security. There are too many IDs and they're too ingrained in our systems to let us change them, even if we wanted to. The problem is, most of the gateways we use do not allow the SMTP e-mail name to differ from the local login ID. There's just no way to specify that in the configuration. In other words, my Internet mail address would have to be **xyz123@calibersys.com**. We found this distasteful for obvious reasons; Actually, it's even worse. The gateways add their network host name to the address for outbound mail. My address would really be **xyz123@gateway1.calibersys.com**. So the first problem to solve is to convert, for outbound messages, the *private* gateway-based addresses to the *public* Internet addresses, for example from **xyz123@gateway1.calibersys.com** to **tlowery@calibersys.com**.

Solving that problem creates another one. The **calibersys.com** e-mail domain is made up of several mail systems. If mail is coming inbound from the Internet to **tlowery@calibersys.com**, how is the decision made to send it to **gateway1** (say, Microsoft Mail) rather than **gateway2** (TAO/Emc2)? Some system along the way has to look up **tlowery** and decide to send the message to **gateway1**. At Caliber, we accept mail for **calibersys.com**, **logistics.calibersys.com**, **shiprps.com**, **vikingfreight.com** and **roberts.com** and perform name lookups for all of them.

Like many companies, our internal TCP/IP network is connected to the Internet via a commercial firewall. We purchased the firewall long before deciding to provide Internet mail service to all the Caliber companies. In a way, it can be thought of as a legacy system with some of its own drawbacks. Although our firewall can solve either the first or second problem, it can't solve both at the same time. According to our firewall vendor, we were their first customer who needed name mapping and gateway hiding for multiple domains, all at the same time. Commercial X.500-based directory systems can be purchased to tie everything together elegantly; the problem is cost. It doesn't make much sense to plunk down $200,000 for a solution that, we hope, won't be needed two years from now. With no quick fix from the firewall vendor in sight and hoards of users beating down my door for inter-company and Internet mail, I sat down and studied sendmail to see if it offered any answers. Luckily it does.

The solution we have in place today can be seen in Figure 1. Any mail message, including those to or from the Internet, that travels from one system to another travels through a central hub. For added reliability, there are actually two hubs. One serves as the primary; the other is a backup in case the primary fails. The sole purpose of these hubs is to hide the e-mail addressing details from users. Let's look at an example.

First, let's say I at **tlowery@calibersys.com** want to send a message to Jane at **jdoe@vikingfreight.com**. Both the sender and recipient addresses exist inside the Caliber firewall, so the message will not be sent to the Internet. My mailbox name on Microsoft Mail is **xyz123** and the SMTP gateway for Microsoft Mail is called **gateway1**. When the message leaves my mail system, it looks like this:

```
From: xyz123@gateway1.calibersys.com
To: jdoe@vikingfreight.com
```

The message is routed to mhub, our primary address mapping hub. First it looks up **xyz123@gateway1.calibersys.com**. When it finds a match, that address is changed to **tlowery@calibersys.com**. It then looks at **jdoe@vikingfreight.com**. There are three possible gateways for **vikingfreight.com**, **gateway1**, **gateway2** and **gateway3**, corresponding to Tandem PS Mail, Microsoft Exchange and TAO/Emc2, respectively. The hub looks up the address and sees that Jane's mailbox (**jdoe**) is on PS Mail and is called **vft1100**. So **mhub** changes

**jdoe@vikingfreight.com** to **vft1100@gateway1.vikingfreight.com**. The message header now looks like this:

```
From: tlowery@calibersys.com
To: vft1100@gateway1.vikingfreight.com
```

The hub then hands the message off to **gateway1.vikingfreight.com** for delivery to Jane's mailbox. Now let's take a look a closer look at the hub.

The primary mail hub is a Compaq Proliant 1500 with a 133 MHz Pentium processor. The backup hub is a Dell OptiPlex with a 75 MHz Pentium. Both are running Linux Slackware version 3.0. The decision to run Linux was an easy one. I wanted a solid, dependable solution at the lowest possible cost—that meant Intel-based hardware and Linux. I've used Linux for various projects over the last three years and have never experienced a kernel crash. I can't say that for all the commercial operating systems I've used. The crucial piece of software for this project is sendmail, available for virtually all Unix variants. I knew if Linux didn't work out, I could swap in a Sun, IBM or HP workstation without having to make software changes. The risk of using Linux as the initial platform was very small. In the four months since the project went live, I'm happy to say that we've had zero problems with it. Now it's time to discuss the nitty-gritty details. Two primary pieces make the hub work: DNS and sendmail. I'll discuss each in turn.

## Enter DNS

What follows is a brief introduction to DNS. If you're already familiar with it feel free to skip the next few paragraphs.

DNS stands for Domain Name System. Its job is to keep track of each computer's name on the network. Programs that communicate with other computers require the numerical address of that computer. If all the program has is a name, it gives that name to DNS and asks for the corresponding address. For example, the mail hub has to get the address for **gateway1.calibersys.com** before it can deliver mail to it. The hub asks DNS for the address and is told something like 11.22.33.44. As soon as the hub has that address, it can contact **gateway1** to deliver the mail. The DNS configuration files are filled with lines like these:

```
mhub.calibersys.com.         IN A 12.34.56.78
mhub2.calibersys.com.        IN A 12.34.66.88
gateway1.vikingfreight.com.  IN A 12.12.12.12
```

The first column is the name the machine goes by. The second column, **IN**, isn't important for our discussion. The third column, **A**, indicates that this is an address record. It just means this line maps a name to an address. The fourth column holds the address of the machine named in column one. In addition to

looking up names and giving back addresses, DNS can also indicate that one computer accepts mail for another. When computer A accepts mail for computer B, A is called a Mail Exchanger for B. Whenever sendmail tries to deliver mail to a given machine, it first looks for a mail exchanger. If no mail exchanger is found, it then looks for a regular address. Let's look at an example:

```
calibersys.com.    IN MX 10 mhub.calibersys.com.
vikingfreight.com. IN MX 10 mhub.calibersys.com.
shiprps.com.       IN MX 10 mhub.calibersys.com.
roberts.com.       IN MX 10 mhub.calibersys.com.
```

These lines tell sendmail that any mail addressed to **calibersys.com**, **vikingfreight.com**, **shiprps.com** or **roberts.com** should be sent to **mhub.calibersys.com**. That's how mail addressed to **jdoe@vikingfreight.com** is routed to the hub. The first column can be thought of as a machine name. There doesn't have to be an actual computer using this name; think of it as a pseudo-machine for e-mail purposes. It's what you would see to the right of the **@** symbol in an e-mail address. Again, we don't care about the **IN** for this discussion. The **MX** in the third column tells DNS that this is a mail exchanger record. Next comes the priority. A machine can have several mail exchangers, each with a different priority. I'll discuss that in a moment. Finally, the last column is the name of the machine acting as the mail exchanger. Any machine acting as a mail exchanger must be a real machine and must have a corresponding address record. Now let's talk about multiple exchangers. Remember that this project involves two hubs, a primary and a secondary, The primary machine is **mhub.calibersys.com**, the secondary is **mhub2.calibersys.com**. Both are listed in DNS. Remember that the number 10 above referred to priority. The lower the number, the higher the priority. Let's say that we see the following lines in the DNS configuration file in addition to the ones above:

```
calibersys.com.   IN MX 20 mhub2.calibersys.com.
vikingfreight.com IN MX 20 mhub2.calibersys.com.
shiprps.com.      IN MX 20 mhub2.calibersys.com.
roberts.com.      IN MX 20 mhub2.calibersys.com.
```

sendmail would first try to send any mail destined for **vikingfreight.com** to **mhub**, since 10 represents a higher priority than 20. If that failed, it would then try to send the mail to **mhub2**, the backup hub. After either hub receives the message, it looks up **jdoe@vikingfreight.com** and converts that address to **vft1100@gateway1.vikingfreight.com**. It would then look up **gateway1.vikingfreight.com**. It does not have a mail exchanger record listed for it, but it does have an address record. This means that **gateway1** accepts its own mail. Using DNS this way to route mail for a domain, e.g., mycompany.com, to an actual computer where the mail is stored, e.g., mail.mycompany.com, is commonplace. What's different here is the address mapping. To see how that's done, we need to look at sendmail.cf.

## On to sendmail

sendmail.cf is the configuration file for sendmail. So what exactly is sendmail? sendmail is the *Swiss Army knife* of mail systems. Officially it's known as a message transfer agent, or MTA. There are a few different flavors; Linux Slackware 3.0 comes with the one known as Berkeley V8. Users typically don't interact with it directly. (That task is left to a mail user agent or MUA such as elm or pine.) sendmail runs in the background, silently routing mail from one computer to another. It was written in the 1970's and 1980's by Eric Allman at U.C. Berkeley. Because of Eric's flexible design, sendmail is still the most widely used MTA on the Internet. It's standard issue software with just about any Unix-based operating system. All that flexibility comes at the price of complexity. sendmail is probably the most complex of all the Unix utilities. I'll cover some of sendmail's features and how they can be used to solve our address mapping problems, but a complete discussion of sendmail is beyond the scope of this article. For more information see the resource box. The Caliber mail hub makes heavy use of three sendmail mechanisms: macros, classes, and database lookups. All these are specified in the configuration file, sendmail.cf. sendmail macros are similar to C language macros. A primary difference is that the macro name can only be one character long. For example,

```
DG gateway1.vikingfreight.com
```

defines the macro **G** as **gateway1.vikingfreight.com**. Its C language equivalent would be:

```
#define G gateway1.vikingfreight.com
```

The macro is invoked later by specifying a dollar sign followed by the macro name, e.g., **$G**. Anywhere the symbol **$G** appears, **gateway1.vikingfreight.com** would be substituted in its place. Classes are very similar to macros. The difference is that they can expand to one of many different values. For example,

```
CU xyz123 abc789 def444
```

defines the class U with three values. Alternatively, sendmail can read the values from an external text file:

```
FU /etc/mail/users
```

This ability is handy if you want to define a class with a large number of values. The class is invoked by specifying a dollar sign, then an equal sign, then the class name, e.g., **$=U**. I'll explain a little later how classes are useful. The third mechanism exploited by the mail hub is the database lookup. sendmail can consult external databases and swap the lookup key with the value found in the

database. A few different database formats are supported; we use GNU dbm databases. The records in these databases have only two fields. The lookup key is the first field. Everything following that is considered a value field. A dbm database with four records could look like this:

```
xyz123  tlowery
ft1100  jdoe
bc789   asmith
def444  bjones
```

If sendmail consults the database looking for **abc789**, it will find the value of **asmith** and substitute that value in place of **abc789** in the e-mail address.

That covers the basic mechanisms. Now let's look at the configuration file itself. sendmail.cf if filled with lines like this one:

```
R$=U@$G  $:$(mapdb $1 $)@$K
```

These lines are known as *rules*. The rules are grouped together in subroutines, each of which is called by sendmail to perform a certain task. These subroutines are called *sets*. It's a terse programming language based on regular expression pattern matching. Each rule examines an e-mail address and may alter it. Rules have two parts, the left-hand-side (LHS) and right-hand-side (RHS), separated by one or more tab characters. The LHS is a pattern; sendmail tries to match the current address with this pattern. If the address matches the pattern, sendmail will rewrite the address based on what the RHS says. If there is no match, the RHS is ignored. Now let's look at the LHS of our sample rule, left to right:

```
R$=U@$G
```

The **R** simply states that this is a rule. All rules start with the letter R. The next three characters, **$=U**, are a class reference. Given the class definition above, **$=U** will match **xyz123**, **abc789** or **def444** successfully. If the address begins with any other string, the match will fail. The next character is a literal **@**. That character must appear in the address for a match to happen. The following two characters, **$G,** are a macro reference. The macro expands to:

```
gateway1.calibersys.com
```

The address **xyz123@gateway1.calibersys.com** would match the pattern and the RHS would be invoked. **tza555@gateway1.calibersys.com** would not match since **tza555** is not a member of the U class. Likewise, **xyz123@gateway2.calibersys.com** would not match since **gateway2.calibersys.com** is not the value of the G macro. Now let's move on to the RHS to see how an address is rewritten. The RHS of our sample rule is this:

```
$:$(mapdb $1 $)@$K
```

The first two characters, **$:** tell sendmail to only invoke this RHS once. By default, sendmail will invoke the RHS repeatedly as long as the result still matches the LHS. Following that is the string,

```
$(mapdb $1 $)
```

which performs the database lookup, tells sendmail to look for a database named **mapdb** and search for the first item from the LHS. **$2** would search for the second item and so on. The first match from our LHS was **xyz123**. sendmail then searches for that string and finds **tlowery**, so it replaces **xyz123** with **tlowery** in the address. The next character in the RHS, **@**, is a literal. It's written to the new address following **tlowery**. Next comes **$K**, a macro reference. Let's assume the macro **K** was defined like so:

```
DK calibersys.com
```

sendmail will place **calibersys.com** after **@** in the new address, completing the rewriting process.

From this we see how a private address of **xyz123@gateway1.calibersys.com** can be converted to a public address of **tlowery@calibersys.com**. The recipient of the message will never know the original sender address was not the public address. Switching from public to private can be accomplished in a similar manner. This discussion of address mapping has only scratched the surface; sendmail's flexibility can help the e-mail administrator solve virtually any mail routing task.

## The Result

The mail routing hub has been in operation at Caliber for about three months, and in that time it has routed over 48,000 messages. Given current traffic statistics, I expect it to handle about 600,000 messages during its first year. Fortunately, the few problems we've experienced have been due to configuration problems. It's too easy to leave out a period here or a dollar sign there. Linux and sendmail have performed flawlessly.

In a perfect world this solution wouldn't be needed. If the Caliber companies used only one common e-mail platform, there would be no need to look up mailbox names and route messages to the right gateway. But many large companies have a number of legacy systems that won't be going away any time soon. Those are exactly the types of environments where tools like sendmail work best. Due to network logistics and geography, we will still be using some of our legacy mail systems until 1998. In the meantime, our Linux e-mail hub

will continue chugging through messages, routing them between disparate platforms and helping us meet increasing user expectations.

Sidebar: DNS and Sendmail Resources

**Tom Lowery** (tlowery@calibersys.com) is Project Manager of E-Mail and Groupware Development at Caliber Technology.

Archive Index Issue Table of Contents

Advanced search

# Java Class Reference Package

**Dave Dittrich**

Issue #30, October 1996

These reference cards are not introductory overviews of the Java language, like the vast majority of Java books on the shelves today.

**Publisher:** Specialized Systems Consultants (SSC)

**ISBN:** 0-916151-95-6

**Price:** $7.00

**Reviewer:** Dave Dittrich

Back in the mid 80s, when I first pulled a Unix workstation—an Intergraph InterPro, running System V Unix—out of a closet at Boeing and began to teach myself Unix system administration, I needed something to help me remember shell command options. The first reference I bought was the System V command reference from Specialized Systems Consultants. It sits in my desk drawer to this day, and still sees daylight every time I'm confronted with the question, “What option is it that I need to use with the **foobar** command on a System V system?” This reference card cost me about $5, if I remember correctly, which is about the same price it goes for today, ten years later.

That SSC has produced a Java class library reference isn't a big surprise. These references are the backbone of their publishing business, which now extends to Linux documentation, CD-ROM software collections, Linux and WWW magazines—even t-shirts! With the new “Internet runs on dog years” world that Java exists in, I seriously doubt if these cards will have the same shelf life as my System V reference, but System V isn't the same as it was back then, either. I'm sure many will find them equally useful just the same.

Before trying to assess these new Java reference cards, I think it's helpful to consider what these references are and are not, and just who is likely to use them.

These reference cards are not introductory overviews of the Java language, like the vast majority of Java books on the shelves today. They do not include tutorials or introductory text for each package, like O'Reilly & Associates' *Java in a Nutshell.* They are not API documentation, like *The Java API* (both volumes) from Addison-Wesley. I don't see them as being "competitors" with anything else out there right now (except, perhaps, the Java API hypertext pages themselves, which are a bit awkward to use sometimes). So what are they, and how well do they do their job?

The Java reference cards—one for the java.applet, java.awt, and java.util packages, and another for the java.lang, java.io, and java.net packages—are a concise, classified (no pun intended) listing of the methods and important constants associated with each class in these packages. No more and no less (well, at least not that much less).

Together, the two references cover 38 panels. Although not stated explicitly, they use a syntax somewhat similar to Unix man pages, where optional parameters are surrounded by square brackets. For example, rather than list two lines for each method signature, like this:

**BufferedInputStream**(InputStream *dest*);**BufferedInputStream**(InputStream *dest*, int *buffersize*);

they include just one line, like this:

**BufferedInputStream**(InputStream *dest* [,int *buffersize*]);

While this syntax is not exactly what you'll find in other books on Java, you get used to it quickly, and most people will probably appreciate the brevity it contributes to the listings.

Each package makes up its own section, with classes within the package in their own graphic box. This makes for a clear delineation between classes within each package. The box titles include only the class name. For example, just the title **Class** heads a box in the **JAVA.LANG** section (I'm not sure why they're **YELLING**), rather than being explicitly labeled **java.lang.Class** as the compiler will expect. In practical use it can be hard to determine exactly what you need to import to use this class in your code. When you are thirteen pages into the card and find the **Color** class, you have to backtrack page by page to find that **Color** is in the **JAVA.AWT** package and know to add **import java.awt.Color;** (or the

more general **import java.awt.*;**) into your code. (This is a pretty minor gripe. If it *really* bothers you, you can always just write in the package name with a pen).

Carefully going through some of the class descriptions also brought out what appear to be a few errors of omission and one parameter mix up. For example, in the **Component** section of **JAVA.AWT**, missing methods are: **checkImage()**, **getPeer()**, **location()**, **prepareImage()**, **size()**, and **toString()**. The **repaint()** method has the *maxWait* parameter at the end of the list, when it should be at the beginning. Since most of these methods involve the complicated image producer/consumer mechanism, or are accessory functions more interesting to people programming layout managers than those just building a simple GUI, the omissions may not matter to the majority of Java coders. Missing from Label is **addNotify()**, but the 1.0.2 JDK API documentation itself says about this method, “Most applications do not call this method directly.”

More glaring is the lack of two entire packages, **java.awt.image** and **java.awt.peer**. Granted, these two packages are more interesting to people doing quite complicated graphics programming, or coding new AWT peers for window managers other than the already supported Windows, Macintosh and Motif, but they are still part of the JDK class library that a programmer may use. The image producer/consumer paradigm is quite confusing and is sometimes criticized as such in Java books, but if the programmer is forced to also have handy a copy of *Java in a Nutshell* to get the whole API picture, many will probably opt to just go with the book.

The author, Randy Chapman, is intimately familiar with the JDK through his work with the Linux port, and I know him from his working days in the Academic Computer Center at the University of Washington to be a very careful and thorough programmer. I am not sure if the omissions are due to working with an older API or if the idea was to simplify things for the average programmer or perhaps just the result of time pressures. (He is, after all, still a student with educational demands high on his priority list. I won't fault him for that.) Since it isn't stated explicitly, I will assume that the aim is to simplify the card and conclude that the target audience will be the beginning to intermediate programmer who sticks to coding “average” Java applications or applets and not the kind of Java fanatics who think triple tall espressos should be purchased in pairs. (These people would prefer to write Perl scripts to extract this information directly from the JDK source code tree and run the results through nroff!)

Overall, these cards are quite handy to have lying next to your keyboard and will prove to be well worth the small price that SSC charges for them. I wish more books had this high a usefulness-to-price ratio.

**Dave Dittrich** (dyittrich@cac.washington.edu) works at the University of Washington in Client Services, Computing & Communications. Visit his web page at: http://www.washington.edu/People/dad/.

Archive Index Issue Table of Contents

Advanced search

# apropos, whatis and makewhatis

**David Bandel**

Issue #30, October 1996

Tools to help you find the appropriate command.

This month's column looks at three very useful and related commands, **apropos**, **whatis** and **makewhatis**. To understand why these commands are so valuable, it helps to grasp the underlying philosophy that continues to guide evolving versions of Unix, including Linux. That philosophy remains one of creating small, portable, specialized programs that perform one task well, and that can receive input from, and redirect output to, other programs.

This philosophy has created a proliferation of small, powerful, but extremely limited programs. Just do a directory listing of /usr/bin, and you'll see what I mean. And that's not all of them. You could sit down and run every one of them to see what they do. Or you could begin reading the hundreds of man pages available describing these commands. No matter which method you use to learn the commands, in the end, you'd still probably forget most of them due to the sheer volume. So how do you know which of the hundreds of programs available can do the job for you? Or which of the commands will be best suited to your particular needs? apropos and whatis come to the rescue.

Apropos, as defined by the dictionary, means “apt; relevant; suited to the occasion, though not strictly belonging to the subject under consideration.” These definitions, particularly the last one, are totally apropos. apropos will list programs with a one line synopsis of each program based on a keyword search. whatis is similar, but even more constrained; i.e., the actual command is given as the argument, rather than a keyword, so there is less output.

## makewhatis

Before we look at how apropos can help us, we need to ensure that the database apropos uses exists and is up-to-date. Enter **makewhatis**. This command creates the whatis database files used by both apropos and whatis.

They are located in each **../man** directory and catalog the manual files in each of the individual cat? and man? subdirectories.

To create the whatis database files, you need to invoke makewhatis as the root user. Non-privileged users normally do not have write permission in the **../man** directories to create the whatis database files. A second clue to the nature of makewhatis is its location. makewhatis is usually found in the **/usr/sbin** subdirectory, indicating its classification as a system administration program. Ordinarily only root's **PATH** environment variable contains the sbin directories. makewhatis may be invoked by root's crontab file and run on a recurring basis, and you may wish to include it if it isn't already there. But that is beyond the scope of this article. If you can log in only as a non-privileged user, or are sure your whatis database files exist, you may want to skip ahead to the next section. If you begin to see **<keyword>: nothing appropriate**, you'll need to have your system administrator run makewhatis.

Running makewhatis for the first time will take several minutes, so be patient (go have a cup of coffee). Run by itself, makewhatis will create the whatis database file in **/usr/man**. To ensure that all the man locations are cataloged, use the **-w** switch. This will read the file **/etc/man.config** and use the man paths it specifies. Or you may add paths following the **-w** switch and they will be used as well. If you are want to know which paths will be cataloged, type **man --path**, and you will see where **/etc/man.config** believes your manuals are. If you have other paths, they should be added to your man.config file.

Another makewhatis option is **-c**. This switch, when used alone, will catalog only the **../man/cat** entries listed in **man.config**. Other cat subdirectories may be added following the **-c** switch, and they will also be cataloged.

You may, however, want only to update the whatis databases with newly added commands. Use the **-u** switch to update the database files. This switch reads the time of the whatis database file and adds those manual pages created or updated since.

If you want to know what makewhatis is doing, add the **-v** switch, and you will see each man directory entered and each command as it is added to the list. Each switch used with makewhatis should be separated by a space and preceded by a hyphen; the switches cannot be combined. makewhatis does have one weakness: if your system does not have sufficient RAM and virtual memory, makewhatis will fail. If you get an error message—and you are running makewhatis as root—add more swap space and try again.

### Using apropos

To search the whatis database on your system, just type:

```
apropos \keyword
```

inserting your criteria as the keyword for the search. Let's try one out. I've never used my CD-ROM player for anything more than mounting a data disk and accessing files, but I'd like to play some music while I'm working (or playing xtetris because it doesn't have music accompaniment, etc.). So I type:

```
apropos cdrom
```

and I see:

```
xplaycd (1)          - X based audio cd player for cdrom drives (END)
```

apropos uses the less pager unless your PAGER environment variable says otherwise. To exit this screen, press q. To scroll, use the up and down arrow keys, or the space bar to go down a screen at a time.

Now I'm not sure, but I believe other programs for accessing the CD exist on my system. Yes, this will work, but how about a choice? Let's try again. This time, I'll try with just CD:

```
apropos cd
```

survey says:

```
Tcl_AsyncCreate, Tcl_AsyncMark, Tcl_AsyncInvoke, Tcl_AsyncDelete
(3) - handle asynchronous events
cd (3)              - Change working directory
curs_window: newwin, delwin, mvwin, subwin, derwin, mvderwin,
dupwin, wsyncup, syncok, wcursyncup, wsyncdown (3) - create curses
windows
eject (1)           - eject CD-ROM disc from drive
mcd (1)             - change MSDOS directory
rexecd (8)          - remote execution server
termios, tcgetattr, tcsetattr, tcsendbreak, tcdrain, tcflush,
tcflow, cfgetospeed, cfget
ispeed, cfsetispeed, cfsetospeed, tcgetpgrp, tcsetpgrp (2)
               - get and set terminal attrib
utes, line control, get and set baud rate, get and set terminal
foreground process group ID
tin, rtin, cdtin, tind (1) - A Netnews reader
cda (1)              Compact disc digital audio player utility
wm2xmcd (1)         - workman-to-xmcd CD database file converter
xmcd (1)            - CD digital audio player utility for X11/Motif
xplaycd (1)         - X based audio cd player for cdrom drives
```

Now we have a problem. We can be overwhelmed with inappropriate items. If this didn't give you a long listing, try giving cat as a keyword for apropos. You'll get pages of output. (Read on to find out why.)

If you look, you can see that we got what we wanted, and a whole lot more. How can we narrow it down? Can we put two keywords on the apropos command line? Yes. Unfortunately, the keywords are logical ORed and not ANDed together, making the output even longer. But if we scan the listing, it

appears most of the commands we're interested in contain the term audio. We could try **"apropos audio"**. But let's search the previous list instead. Type:

```
apropos cd | grep -i audio
```

Our reward:

```
cda (1)  - Compact disc digital audio player
utility xmcd (1 - CD digital audio player utility
for X11/Motif xplaycd (1)     - X based audio cd
player for cdrom drives
```

Now, that's more like it! We can try these programs to see which we like best. And we know they have manual pages to help us out.

A good exercise for the reader might be to use “mail” as a keyword (this will return a very long list), then grep the list for audio to see which programs might help you e-mail sound files.

A slightly less obvious, but identical command for apropos exists. The command **man -k <keyword>** is synonymous, though not as mnemonic.

### whatis

Finally, let's take a look at whatis. We've been accessing the whatis database files during the apropos (aka man **-k**) searches. Let's try our cd search using whatis instead.

```
whatis cd
```

Now we get:

```
cd (3)               - Change working directory
```

So what's the difference? Why only one entry? Think of the whatis database as columnar and containing two columns. The left column contains the program name (the command used to invoke the program) and the right side contains the first line of the manual's program synopsis. apropos searches both columns using the keyword as a regular expression to find all occurrences of the keyword. These occurrences may be embedded in the command word or the words of the synopsis. For example, **apropos cat** returns lines containing the word catalog, category, duplicate, application, etc. whatis, on the other hand, searches only the left hand column, which contains only the program name. This feature is helpful if you know the name of a command, but not its function.

## Drawbacks

These commands do have limitations. If a command has no corresponding manual page, it will not be listed in the database. makewhatis does not include a manual page, at least not on the author's system. If a synopsis does not contain a keyword you have chosen to search on, it will not show up. As with all tools, you may need to compare the results of several different searches or **grep** long search results to find the best program for your needs.

## Summary

Now you have the tools to help find commands you may not use often, but are worthwhile knowing. Using apropos and whatis sure beats reading through all the manual pages. A newbie to Linux will find a lot of directory manipulation help with the dir keyword. So put apropos to work and search and learn!

**David Bandel** (dbandel@ix.netcom.com) is a Computer Network Consultant specializing in Linux, but begrudgingly works with Windows and those “real” Unix boxes like DEC 5000s and Suns. When he's not working, he can be found hacking his own system or enjoying the view of Seattle from 2,500 feet up in an airplane. He welcomes your comments, criticisms, witticisms, and will be happy to further obfuscate the issue.

Archive Index Issue Table of Contents

Advanced search

# New Products

**LJ Staff**

Issue #30, October 1996

Java Generic Library, Web ToolKit, InvisibleWeb & Offline Proxy Server and more.

## Java Generic Library

ObjectSpace,Inc. announced its Java Generic Library (JGL) is now available free for commercial use. JGL is a comprehensive set of 14 reusable containers and 70 algorithms for Java and is completely compatible with Sun Microsystems' Java Developer's Kit (JDK). The release includes full source code, on-line HTML documentation, examples, tutorial and a suite of performance benchmarks. JGL can be downloaded from www.objectspace.com.

Contact: ObjectSpace, Inc. 14881 Quorum Drive, Suite 400, Dallas, TX 75240, Phone: 214-934-2496, FAX: 214-663-9100, E-mail: jgl@objectspace.com, URL: www.objectspace.com.

## Web ToolKit

ObjectSpace, Inc. also announced the release of Web<ToolKit>, a new ANSI/ISO compatible C++ class library for producing dynamic and interactive HTML World Wide Web pages. It supports HTML page creation using a set of C++ classes representing HTML elements, including test, links, graphics, tables, forms, frames and widgets. Software for UNIX platforms will be available for a cost of $475.

Contact: ObjectSpace, Inc., 14811 Quorum Dr., Suite 400, Dallas, TX 75240, Phone: 214-934-2496, Fax: 214-663-9100, E-mail: info@objectspace.com, URL: www.objectspace.com.

### InvisibleWeb & Offline Proxy Server

Innovative Software now offers the first two Linux ports of products for Internet users. The first port is the Offline Proxy Server that allows the user to switch between an “online” and “offline mode.” When in offline mode, it doesn't try to contact remote servers when a document is available on the local disk; in online mode it is transparent. The second port is the InvisibleWeb, a command-line tool that allows you to automate the process of downloading documents from a WWW site. The two products can be used separately or together. Free evaluation copies of both products can be downloaded from: www.isg.de/visualweb/invisible_e.html.

Contact: Innovative Software GmbH, Kaiserstrasse 65, 60329 Frankfurt am Main, Phone: +49-69-236929, Fax: +49-69-236930, E-mail: lkv@barbar.isg.de URL: www.isg.de/visualweb.

### Internet and Intranet/Web Server with Cyrix 166MHz chip

Datacomm Technologies announced the availability of Internet/Intranet Servers, Interserve-6150 and 6166, constructed using fast 6x86 CPU(s) from Cyrix Corporation, a Texas chip manufacturer. Interserve-6150 and 6166 are multi-user/multi-tasking web servers with a true 32-bit operating sytem. For the Internet both servers provide Netscape Navigator 2.0, complete integration of DNS/NIS, HTTP, SMTP, FTP, SNMP, PPP/SLIP, NNTP, WAIS, Gopher and more. For Intranet services both servers provide the ability to act as Netware 3.x/4.x client, with auto-mounted volumes, Enterprise wide bakup utilty and printing, drag & drop administration tools and more. These servers run under Linux using Caldera.

Contact: Datacomm Technologies, 1617 S. Norton Ave., Los Angeles, CA, 90019, Phone: 800-607-9640, 213-737-5599, E-mail: micro_x@earthlink.net

### ARDI Executor 2

ARDI announced the release of Executor 2 on CD-ROM now in beta release. The CD contains Executor/DOS, Executor/Linux and Executor/NEXTSTEP as well as a sample of Macintosh freeware, demoware and shareware. Executor 2 allows serious Macintosh applications and games to run under Linux. During the beta testing, the Executor2 CD is available for $149. Beta CD licensees will automatically be sent the complete Executor 2 CD when it is ready. Price of finished release will be $249.

Contact: ARDI, 1650 University Blvd. NE, Suite 4-101, Albuquerque, NM 87102, Phone: 505-766-9115, FAX: 505-766-5153, E-mail: info@ardi.com , URL: www.ardi.com.

### TowerEiffel Release 2.0

Tower Technology Corporation announced the availability of TowerEiffel Release 2.0. TowerEiffel is a full object-oriented life-cycle development environment for building reusable frameworks, applications and systems. It is available for many Unix platforms, including Linux. Two versions will be available: the Professional version at a price of $995 for Linux, and an introductory Lite version for a price of $325.

Contact: Tower Technology Corp., Austin, Texas, Phone: 512-452-9455, E-mail: may@twr.com , URL: www.twr.com.

### Debian Linux 1.1

Software in the Public Interest announced the release of Debian Linux 1.1, a free-software Linux system. The Debian 1.1 system includes 474 software packages, including the Linux 2.0 kernel and all-ELF executables. A complete list with descriptions can be found at www.debian.org/debian/FTP. Debian allows the entire system, or any individual component, to be up-graded in place without reformatting, without losing custom configuration files, and (in most cases) without rebooting the system. Debian Linux 1.1 can be retrieved from ftp://ftp.debian.org/debian/Debian-1.1/.

Contact: Software in the Public Interest, E-mail: bruce@pixar.com, URL: www.debian.org.

### COS/Print

OSM (Open Systems Management) announced the release of COS/Print, a network-aware spooling and printer management package for Unix platforms including Linux. COS/Print provides full control over print jobs, queues and printers, plus multiple levels of security. It is available as a stand-alone product for $400 or as part of OSM's COSMOS systems management printer package.

Contact: Open Systems Management Inc., 1111 Third Avenue, Suite 2500, Seattle, WA 98101, Phone: 206-583-8373, Fax: 206-292-4965, E-mail: mike@osminc.com

### Tecplot 7.0

Amtec Engineering has announced the release of Tecplot version 7.0. Tecplot provides engineers and scientists with the broadest set of tools available for visualizing and plotting large amounts of data. This new release features a new graphical user interface, animation and page layout. Tekplot offers a wide variety of viewing options, including wire-mesh plots, contour lines, vector

fields and XY plots, all easily annotated and customized. Tecplot runs on most Unix workstations (under Motif) including Linux. Single-user pricing ranges from $995 to $3,195, depending on the platform and the license type.

Contact: Amtec Engineering, Inc., P.O.Box 3633, Bellevue,WA 98009-3633, Phone: 206-827-3304, Fax: 206-827-3989, E-mail: mike@amtec.com , URL: www.amtec.com.

### VanillaSearch

Thought Inc. announced the release of their new product VanillaSearch, a new Java based pattern matching search class derived from Perl and grep syntax. VanillaSearch provides an extremely powerful unicode pattern-matching search capability, using industry standard English or programmer defined foreign language META characters. VanillaSearch is available for students and non-commercial use for $49.00; for commercial use for $495 for the binary code only and $995 for both binary and source code.

Contact: Thought Inc., 2222 Leavenworth St. Suite 304, San Francisco, CA 94133. Phone: 415-928-4229, FAX: 415-567-9945, E-mail: info@thoughtinc.com, URL: www.thoughtinc.com

### Liquid Reality Developers Kit

Dimension X has released a beta version of their Liquid Reality developers kit. Liquid Reality is the first platform independent implementation of VRML 2.0 coded entirely in Java. The toolkit includes support for 3-D sound, compatibility with multi-user servers, an open API, 250 classes to support 3-D content creation, Java classes, access to ICE, a low level high speed 3D graphics engine and the ability to create VRML2.0 applets that do not have to be installed on the desktop. The beta version can be downloaded free from the Dimension X web site, www.dimensionx.com.

Contact: Dimension X, 181 Fremont St., Ste. 120, San Francisco, CA 94105, Phone: 415-243-0900, E-mail: megan@dimensionx.com, URL: www.dimensionx.com.

### Accelerated OpenGL Solution for Linux

X inside, Inc. announced a software implementation of OpenGL for Linux. OpenGL is a software package developed by Silicon Graphics for the rendering of 3D objects. Product features include support for seven operating systems, over 400 graphics cards and chip sets, 4bpp and 8bpp color index mode, support for 8bpp, 15bpp, 15bpp and 224bpp rgb mode, direct rendering to the

frame buffer and loadable X-server extension to Accelerated X. This product is available for under $350 per seat.

Contact: X Inside, Inc., 1801 Broadway, Suite 1710, Denver, Colorado 80202, Phone: 800-946-7433, Fax: 303-298-1406, E-mail: sales@xinside.com , URL: www.xinside.com.

### Tactician Plus

Spire Technolgies is now a distributor for Tactician Plus. Tactician Plus lets you query multiple databases, analyze data and display results in striking graphics. It has a consistent user interface across DOS, Unix and VMS environments, supporting both character and Motif GUI for X-terminals. Spire also distributes WordPerfect for Linux, a spreadsheet, a firewall server and backup/restore software solutions. For price information, contact Spire directly.

Contact: Spire Technologies, Inc., 311 N. State St., P.O.Box 1970, Orem, Utah 84059, Phone: 801-226-3355, Fax: 801-224-3847, URL: www.spire.com.

Archive Index Issue Table of Contents

Advanced search

*Consultants Directory*

> This is a collection of all the consultant listings printed in *LJ* 1996. For listings which changed during that period, we used the version most recently printed. The contact information is left as it was printed, and may be out of date.

**ACAY Network Computing Pty Ltd**
Australian-based consulting firm specializing in: Turnkey Internet solutions, firewall configuration and administration, Internet connectivity, installation and support for CISCO routers and Linux.

Address:
Suite 4/77 Albert Avenue, Chatswood, NSW, 2067, Australia
+61-2-411-7340, FAX: +61-2-411-7325
sales@acay.com.au
http://www.acay.com.au

**Aegis Information Systems, Inc.**
Specializing in: System Integration, Installation, Administration, Programming, and Networking on multiple Operating System platforms.

Address:
PO Box 730, Hicksville, New York 11802-0730
800-AEGIS-00, FAX: 800-AIS-1216
info@aegisinfosys.com
http://www.aegisinfosys.com/

**American Group Workflow Automation**
Certified Microsoft Professional, LanServer, Netware and UnixWare Engineer on staff. Caldera Business Partner, firewalls, pre-configured systems, world-wide travel and/or consulting. MS-Windows with Linux.

Address:
West Coast: PO Box 77551, Seattle, WA 98177-0551
206-363-0459
East Coast: 3422 Old Capitol Trail, Suite 1068, Wilmington, DE 19808-6192
302-996-3204
amergrp@amer-grp.com
http://www.amer-grp.com

**Bitbybit Information Systems**
Development, consulting, installation, scheduling systems, database interoperability.

Address:
Radex Complex, Kluyverweg 2A, 2629 HT Delft, The Netherlands
+31-(0)-15-2682569, FAX: +31-(0)-15-2682530
info@bitbybit-is.nl

**Celestial Systems Design**
General Unix consulting, Internet connectivity, Linux, and Caldera Network Desktop sales, installation and support.

Address:
60 Pine Ave W #407, Montréal, Quebec, Canada H2W 1R2
514-282-1218, FAX 514-282-1218
cdsi@consultan.com

**CIBER*NET**
General Unix/Linux consulting, network connectivity, support, porting and web development.

Address:
Derqui 47, 5501 Godoy Cruz, Mendoza, Argentina
22-2492
afernand@planet.losandes.com.ar

**Cosmos Engineering**
Linux consulting, installation and system administration. Internet connectivity and WWW programming. Netware and Windows NT integration.

Address:
213-930-2540, FAX: 213-930-1393
76244.2406@compuserv.com

**Ian T. Zimmerman**
Linux consulting.

Address:
PO Box 13445, Berkeley, CA 94712
510-528-0800-x19
itz@rahul.net

**InfoMagic, Inc.**
Technical Support; Installation & Setup; Network Configuration; Remote System Administration; Internet Connectivity.

Address:
PO Box 30370, Flagstaff, AZ 86003-0370

602-526-9852, FAX: 602-526-9573
support@infomagic.com

**Insync Design**
Software engineering in C/C++, project management, scientific programming, virtual teamwork.

Address:
10131 S East Torch Lake Dr, Alden MI 49612
616-331-6688, FAX: 616-331-6608
insync@ix.netcom.com

**Internet Systems and Services, Inc.**
Linux/Unix large system integration & design, TCP/IP network management, global routing & Internet information services.

Address:
Washington, DC-NY area,
703-222-4243
bass@silkroad.com
http://www.silkroad.com/

**Kimbrell Consulting**
Product/Project Manager specializing in Unix/Linux/SunOS/Solaris/AIX/ HPUX installation, management, porting/software development including: graphics adaptor device drivers, web server configuration, web page development.

Address:
321 Regatta Ct, Austin, TX 78734
kimbrell@bga.com

**Linux Consulting / Lu & Lu**
Linux installation, administration, programming, and networking with IBM RS/6000, HP-UX, SunOS, and Linux.

Address:
Houston, TX and Baltimore, MD
713-466-3696, FAX: 713-466-3654
fanlu@informix.com
plu@condor.cs.jhu.edu

**Linux Consulting / Scott Barker**
Linux installation, system administration, network administration, internet connectivity and technical support.

Address:
Calgary, AB, Canada
403-285-0696, 403-285-1399
sbarker@galileo.cuug.ab.ca

**LOD Communications, Inc**
Linux, SunOS, Solaris technical support/troubleshooting. System installation, configuration. Internet consulting: installation, configuration for networking hardware/software. WWW server, virtual domain configuration. Unix Security consulting.

Address:
1095 Ocala Road, Tallahassee, FL 32304
800-446-7420
support@lod.com
http://www.lod.com/

**Media Consultores**
Linux Intranet and Internet solutions, including Web page design and database integration.

Address:
Rua Jose Regio 176-Mindelo, 4480 Cila do Conde, Portugal
351-52-671-591, FAX: 351-52-672-431
http://www.clubenet.com/media/index.html/

**Perlin & Associates**
General Unix consulting, Internet connectivity, Linux installation, support, porting.

Address:
1902 N 44th St, Seattle, WA 98103
206-634-0186
davep@nanosoft.com

**R.J. Matter & Associates**
Barcode printing solutions for Linux/UNIX. Royalty-free C source code and binaries for Epson and HP Series II compatible printers.

Address:
PO Box 9042, Highland, IN 46322-9042
219-845-5247
71021.2654@compuserve.com

**RTX Services/William Wallace**
Tcl/Tk GUI development, real-time, C/C++ software development.

Address:
101 Longmeadow Dr, Coppell, TX 75109
214-462-7237
rtxserv@metronet.com
http://www.metronet.com/~rtserv/

**Spano Net Solutions**
Network solutions including configuration, WWW, security, remote

system administration, upkeep, planning and general Unix consulting. Reasonable rates, high quality customer service. Free estimates.

Address:
846 E Walnut #268, Grapevine, TX 76051
817-421-4649
jeff@dfw.net

**Systems Enhancements Consulting**
Free technical support on most Operating Systems; Linux installation; system administration, network administration, remote system administration, internet connectivity, web server configuration and integration solutions.

Address:
PO Box 298, 3128 Walton Blvd, Rochester Hills, MI 48309
810-373-7518, FAX: 818-617-9818
mlhendri@oakland.edu

**tummy.com, ltd.**
Linux consulting and software development.

Address:
Suite 807, 300 South 16th Street, Omaha NE 68102
402-344-4426, FAX: 402-341-7119
xvscan@tummy.com
http://www.tummy.com/

**VirtuMall, Inc.**
Full-service interactive and WWW Programming, Consulting, and Development firm. Develops high-end CGI Scripting, Graphic Design, and Interactive features for WWW sites of all needs.

Address:
930 Massachusetts Ave, Cambridge, MA 02139
800-862-5596, 617-497-8006, FAX: 617-492-0486
comments@virtumall.com

**William F. Rousseau**
Unix/Linux and TCP/IP network consulting, C/C++ programming, web pages, and CGI scripts.

Address:
San Francisco Bay Area
510-455-8008, FAX: 510-455-8008
rousseau@aimnet.com

**Zei Software**
Experienced senior project managers. Linux/Unix/Critical business software development; C, C++, Motif, Sybase, Internet connectivity.

Address:
2713 Route 23, Newfoundland, NJ 07435
201-208-8800, FAX: 201-208-1888
art@zei.com

Archive Index Issue Table of Contents

Advanced search

### *Announcing the Linux Buyer's Guide*

With the Linux market growing at such a phenomenal rate, it is often hard to locate the Linux product you are looking for. Often Linux users aren't even aware of many of the products available for Linux. The solution? *Linux Journal's* annual *Linux Buyer's Guide*.

The 1997 *Linux Buyer's Guide* will be a complete listing of Linux hardware, software and reference materials. With easy-to-read cross-referencing, complete listings of all Linux vendors, and helpful Linux buying information, the *Linux Buyer's Guide* will be an excellent resource for all Linux users.

The annual *Linux Buyer's Guide* will be a free thirteenth issue to all *Linux Journal* subscribers. If you are not yet subscribed, make sure to do so now to ensure the arrival of your copy of the 1997 *Linux Buyer's Guide*. (E-mail info@linuxjournal.com or call 206-782-7733 for *Linux Journal* subscription information.) The 1997 *Linux Buyer's Guide* will have a newsstand date of late February 1997.

If you have a product or service you would like listed in the 1997 *Linux Buyer's Guide*, please call 206-782-7733 and request that a free listing form be sent to you. Or, visit our *Linux Buyer's Guide* WWW site: http://www.ssc.com/lj/guide.html.