Linux Journal Issue #99/July 2002

Features

autoSql and autoXml: Code Generators from the Genome Project *by Jim Kent and Heidi Brumbaugh*

When working with large amounts of data, save yourself some time with these tools, developed as a result of work on the Genome Project.

Multiheading Linux Systems *by Brian Gollsneider and Arthur M. Messenger*

Get set up for those jobs that require double the screen space.

Icarus Verilog: Open-Source Verilog More Than a Year Later *by Stephen Williams and Michael Baxter*

More competitive than ever—and still free!

Interview

A Conversation with Stephen Williams *by Michael Baxter*

Stephen reveals the secrets of just how Icarus Verilog has achieved amelioration.

Indepth

Keeping Up with Python: the 2.2 Release *by Wesley J. Chun*

Unification, iterators and more—the improvements to the Python 2.2 release series.

DSI: Secure Carrier-Class Linux *by The DSI Team*

Security architecture specifically for clustered environments is lacking—but that will soon change.

LINUX JOURNAL

autoSql and autoXml: Code Generators from the Genome Project

Jim Kent

Heidi Brumbaugh

Issue #99, July 2002

These tools have saved us from the drudger of writing tens of thousands of lines of repetitive code—we hope you find them useful.

Moving data from one source to another is not all that difficult in the scheme of things. If your source data is a tab-delimited file, for example, and you need it in an SQL relational database, you might write a little SQL definition, then churn out a C program to read the data from the source file and write it out to the database. But when you're dealing with a big project, or in our case, really big, and you find yourself working with dozens of sources giving you gigabytes of data, writing all that code gets old fast.

To solve this problem, here are two tools to do the job. Together, they generate database definitions for SQL, write C header files with your data definitions and function prototypes, write C code to get data to and from C structures and generate C code for an XML parser.

The Human Genome Browser

The human genome is the instruction manual that is encoded in our DNA. It is made up of three billion pairs of chemical letters, commonly known by the initials G, C, A and T. The genome data is 24 long strands of these letters—not exactly light reading. The Human Genome Browser is a web site at the University of California, Santa Cruz that gives scientists around the world a visual representation of this mountain of data. The browser combines the sequence data itself with higher-level annotations of the function of particular regions of the genome. Users can locate and zoom in on genes they are interested in, link to research conducted on that section of the genome and

compare the genomic data with that of other species. The browser stacks particular types of annotations as *tracks* beneath genome coordinate positions.

The Genome Browser has an HTML/CGI front end that lets the user view and (with the help of dynamically generated image maps) click on genome tracks. Form fields give the user a way to set zoom level and control the data density of the tracks. The CGI source code is C, and the genome data is stored in an SQL database.

There is a lot of data. The browser source code has to deal with data formats for gene prediction and for similarities between the human genome and the genomes of other species. Complicating matters is the fact that we collaborate with at least a dozen external sources that each have data in their own format. Even if we don't want to use their data formats internally, we still need to write a parser to read them in and convert them to our own format. Probably half of our use of the code generators is to make it easier to import files from other groups.

Enter autoSql

autoSql generates SQL and C code for saving and loading data to a database. By using autoSql, we don't need to write the tedious data definition, which involves reading and writing code. For example, the browser has around 30 public tracks and 30 experimental tracks. Each track is associated with a table in a relational database. All of the modules that load a track table into memory are generated by autoSql.

The XML Connection

Later in the project, we started using XML to collaborate with a research group in Japan. XML is also useful to exchange data with other public sites via DAS (the Distributed Annotation System, a protocol for transferring genomic data over the Internet).

autoXml generates C code for an XML parser given an XML DTD file. Since XML I/O is even more code intensive than SQL I/O, autoXml has already proven to be useful.

Worth Their Weight in Coding

Together, autoSql and autoXml have proved to be invaluable time-savers. **autoSql** has been a critical workhorse to the browser project. At 1,200 lines, it has generated fully half the browser program, tens of thousands of lines of code.

Although we don't use XML as much as SQL, we've already broken even with autoXml. In a single project to import data from the Riken mouse genome annotation project in Japan, autoXml generated approximately 1,500 lines of code. (It's only 1,200 lines itself.)

You can download the binaries for autoSql and autoXml from www.soe.ucsc.edu/~kent/exe. The source code, Linux executables and examples from this article are at www.soe.ucsc.edu/~kent/src/autoCode.tgz.

autoSql Overview

autoSql is a program that automatically generates an SQL table creation script and C code for saving and loading data to a database based on an object specification. (See Figure 1 for an overview of this process.)

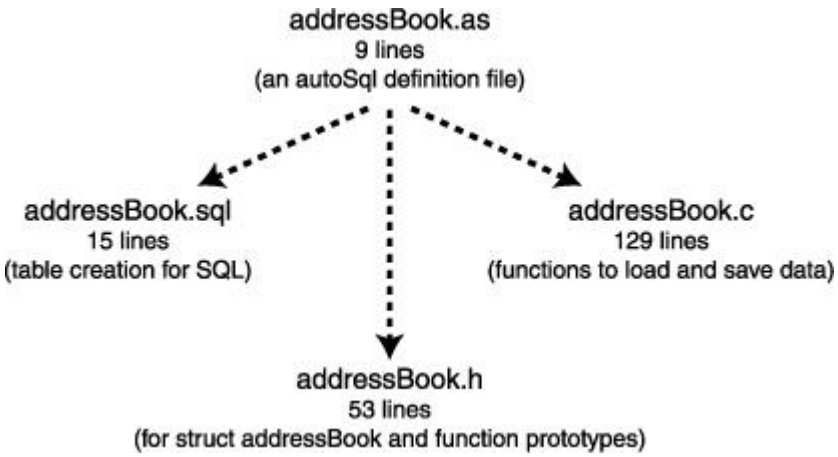


Figure 1. Running a Definition File through autoSql

The specification language is a bit quirky, but it has proven effective for many jobs. We originally developed autoSql for use with a relational database; it turns out that it generates code that can load from many flat formats as well, as long as they are in a text format.

A Simple Example

Imagine a simple address book that just stores name, street address, zip code and state. The autoSql specification for this would be:

```
table addressBook
"A simple address book"
   (
   string name;
     "Name - first, last, both, we don't care"
   lstring address;  "Street address"
   string city;  "City"
   uint zipCode; "A zip code
     is always positive, so can be unsigned"
   char[2] state;
     "Just store the abbreviation for the state"
   )
```

If this looks like a bit of a hybrid between a C structure and an SQL table definition, it's because Jim was switching between C and SQL when he made the autoSql language.

When you run the address book template through autoSql, the program produces the SQL table definition:

```
    #A simple address book
    CREATE TABLE addressBook (
    name varchar(255) not null,    # Name -
        first, last, both, we don't care
    address longblob not null,     # Street address
    city varchar(255) not null,    # City
    zipCode int unsigned not null, # A zip code is
        always positive, so can be unsigned
    state char(2) not null,        # Just store
        the abbreviation for the state
                    #Indices
    PRIMARY KEY(name)
     );
```

and the following C structure definition:

```
  struct addressBook
  /* A simple address book */
      {
      struct addressBook *next;
        /* Next in singly linked list. */
      char *name;
        /* Name - first, last, both, we don't care */
      char *address;    /* Street address */
      char *city;       /* City */
      unsigned zipCode; /* A zip code is always
        positive, so can be unsigned */
      char state[3];    /* Just store
        the abbreviation for the state */
      };
```

Typically in C you access a single row of an SQL database at a time. The row is returned as an array of strings. It is up to the C program to convert the ASCII representation of numbers to binary numbers. This is not hard work, but after you've typed in 20 or 30 lines that look something like:

```
    point->x = atoi(row[1]);
    point->y = atoi(row[2]);
```

you'll appreciate the following two routines that autoSql generates for you:

```
     void addressBookStaticLoad(char **row,
           struct addressBook *ret);
     /* Load a row from addressBook table into ret. */
     /* The contents of ret will be replaced at the */
     /* next call to this function. */
     struct addressBook *addressBookLoad(char **row);
     /* Load a addressBook from row fetched with */
     /* select * from addressBook from database.*/
     /* Dispose of this with addressBookFree(). */
```

The first routine typically is used when you just want to process one item at a time. It doesn't allocate any dynamic memory, and so it's quite fast. The second routine saves the structure to dynamic memory. Since the C structure always

includes a “next” field, you easily use this routine to build a list of address book entries.

The only problem with using dynamic memory is that you have to remember to free it. While autoSql can't remember to free things for you, it can generate routines to free a single dynamically allocated structure, or a list of dynamically allocated structures. That's what the next two routines do:

```
void addressBookFree(struct addressBook **pEl);
/* Free a single dynamically allocated
 * addressBook such as created with
 * addressBookLoad(). */
void addressBookFreeList(
   struct addressBook **pList);
/* Free a list of dynamically
 * allocated addressBook's */
```

Reading structures without having to write code to load them up a field at a time is nice, but sometimes you need to write structures, too. **autoSql** assumes that you'll either want to save the structure in a tab-delimited or in a comma-delimited format. It generates a routine that can do either:

```
void addressBookOutput(struct addressBook *el,
   FILE *f, char sep, char lastSep);
/* Print out addressBook. Separate fields with
 * sep. Follow last field with lastSep. */
```

and macros that make it convenient to do commas or tabs:

```
#define addressBookTabOut(el,f)
  addressBookOutput(el,f,'\t','\\n');
/* Print out addressBook as a line in a
 * tab-separated file. */
#define addressBookCommaOut(el,f)
  addressBookOutput(el,f,',',',');
/* Print out addressBook as a comma
 * separated list including final comma. */
```

autoSql generates a routine to read comma-separated lists. While you are unlikely to call this routine directly yourself, fields more complicated than simple strings or integers get saved in the database as comma-separated lists. This routine allows autoSql to have objects that contain other objects.

Types of Objects

autoSql has three types of objects:

- Simple: objects that contain no variable-sized arrays.
- Object: objects that can contain variable-sized arrays. A next pointer is automatically inserted as the first field in the C structure corresponding to an object.
- Table: like objects, but the program generates an SQL as well as a C definition.

Simple objects differ from other objects in how the program treats array declarations. In the field declaration:

```
simple point[3] triangle;  "A three sided figure"
```

the three points are stored in memory as a C array. If this were declared instead as

```
object point[3] triangle;  "A three sided figure"
```

the three points would be stored in memory as a singly linked list.

Types of Fields

The following basic field types are supported:

- int: 32-bit signed integer
- uint: 32-bit unsigned integer
- short: 16-bit signed integer
- ushort: 16-bit unsigned integer
- byte: 8-bit signed integer
- ubyte: 8-bit unsigned integer
- float: single precision IEEE floating point
- char: 8-bit character (can only be used in an array)
- string: variable length string up to 255 bytes long
- lstring: variable length string up to 2 billion bytes long

Additionally, the simple, object and table types can be used as fields.

Fixed Length and Variable Array Declarations

An array can be declared as either fixed size or variable size. A variable sized array is declared by putting a field name inside of the brackets in the array declaration. This field must be defined before the array.

A More Complicated Example

Imagine that you've just built an amazing 3-D modeling program. The only problem is that now you need to save the structures in a database. Listing 1 is a way you might build the database with autoSql. Saving it as threeD.as and running

```
autoSql threeD.as threeD
```

would end up generating 393 lines of bug-free (I think!) C code and 14 lines of SQL for the investment of 33 lines of specification. (Refer to Listing 2 for the complete autoSql grammar.)

Listing 1. Building the Database with autoSql

Listing 2. autoSql Grammar

autoXml Overview

autoXml generates C code for an XML parser given an XML DTD file. It will generate a structure for each “element” in the DTD and populate the structure with fields for each attribute of the structure. By default, it will generate a parser that ignores elements and attributes not in the DTD, but otherwise is a validating parser. If you use the -picky flag, it will be fully validating.

The autoXml parser will load the entire file into memory. If this is a problem you'll have to resort to the lower-level xap parser, which is much like the commonly used expat parser, but a bit faster.

A Short XML and DTD Tutorial

If you find yourself befuddled by all the acronyms so far, you're probably new to XML (eXtensible Markup Language). It has a tag-based format, and a simple example of an XML doc might be:

```
<POLYGON id="square">
   <DESCRIPTION>
      This is soooo square man
   </DESCRIPTION>
   <POINT x="0" y="0" ->
   <POINT x="0" y="1" ->
   <POINT x="1" y="1" ->
   <POINT x="1" y="0" ->
</POLYGON>
```

Everything in XML lives between <TAG></TAG> pairs. A tag may have associated text, attributes and subtags. In the example above, POLYGON has the subtags DESCRIPTION and POINT, the attribute id and no text. DESCRIPTION has the text “This is soooo square man” and no subtags or attributes. POINT has the attributes x and y. POINT also illustrates a little XML shortcut: tags containing only attributes can be written <TAG att=“something” -> as a shortcut for <TAG att=“something”></TAG>.

XML is much like HTML but has significant differences. All attributes must be enclosed in quotes in XML, while quotes are optional in HTML. Tags must strictly nest in XML, while HTML allows tags to be opened but not closed. The tags in HTML are predefined. In XML the definition of tags is up to you.

Tags can be defined two ways in XML: by a DTD file or by an XML schema. There are pros and cons for each method. DTD files are relatively simple and are recognized by a wide variety of parsers and XML browsers. On the other hand, DTD files can't express that a certain attribute has to be numerical. XML schemas are more complex. They are themselves written in a type of XML, which is nice in some ways. They are not as widely supported yet. Currently autoXml only works with DTD files with some modest extensions.

Here is a DTD file that would describe the POLYGON format above:

```
<!ELEMENT POLYGON (DESCRIPTION? POINT+)>
<!ATTLIST POLYGON id CDATA #REQUIRED>
<!ELEMENT DESCRIPTION (#PCDATA)>
<!ELEMENT POINT>
<!ATTLIST POINT x CDATA #REQUIRED>
<!ATTLIST POINT y CDATA #REQUIRED>
<!ATTLIST POINT z CDATA "0">
```

The DTD has two major types of definitions: ELEMENTs and ATTLISTs (or attributes). An element definition includes the name of the element and an optional parenthesized list of sub-elements. The sub-elements must be defined elsewhere in the DTD with the exception of the #PCDATA sub-element, which is used to indicate that the element can have text between its tags. Each sub-element may be followed by one of the following characters:

- ?: the sub-element is optional.
- +: the sub-element occurs at least once.
- *: the sub-element occurs 0 or more times.

If there is no following character the sub-element occurs exactly once.

The ATTLIST defines an attribute and associates it with an element. It is good style to keep ATTLISTs together with their ELEMENT. Here are the fields in an ATTLIST:

- element: name of element this is associated with.
- name: name of this attribute.
- type: generally CDATA. Can be a reference or date, but these are not supported by autoXml.
- default: this contains a default value to be used if the attribute is not present. The keyword #REQUIRED in this field means that the attribute must be present. The keyword #IMPLIED means that it's okay for this attribute to be missing (in which case it will have a NULL or zero value after it is read by autoXml).

autoXml Extensions and Limits

autoXml extends the type field of ATTLIST to include INT or FLOAT for numerical rather than string values. Similarly you can use #INT or #FLOAT in place of #PCDATA to put a numerical type in the text field. If you include these extensions, please use the .dtdx rather than .dtd suffix on your DTD file.

Currently autoXml only copes with DTD comments if they start on a line by themselves. **autoXml** expects all ELEMENTS and ATTLIST declarations to fit on a single line. It doesn't handle reference data types beyond saving the reference ID as a string.

Listing 3. autoXml Code Generation

Refer to Listing 3 for a complete example of the source code autoXml generates. In addition to the .h file shown in Listing 3, autoXml generates a corresponding .c file as well. Each XML file has to have a root object. In this case the root object is POLYGON (our DTD as is won't let us have more than one polygon per file). You can read an XML file that respects this DTD using the polyPolygonLoad() function, and save it back out using the polyPolygonSave.

autoSql and autoXml work well on a range of data, as you've seen, anywhere from an address book to gene tracks. We hope you'll find these tools useful on your own projects.

Resources

Jim Kent, PhD, and his work on the Human Genome Project have been profiled in the New York Times, the *San Francisco Chronicle*, *Software Development* magazine and other publications. He is currently working on cross-species genomic comparisons and Parasol, a job controller for his kilocluster.

Heidi Brumbaugh (heidi_b@pacbell.net) has been a writer and editor in the computer publishing industry since the late eighties. Visit links to her projects and read some of her fiction at www.heidi.to.

LINUX JOURNAL

Multiheading Linux Systems

Brian Gollsneider

Arthur M. Messenger

Issue #99, July 2002

How to use otherwise obsolete hardware to get a working Linux system on multiple displays.

Engineering today requires a lot of screen real estate. When you start doing circuit layouts, math models and simulations, image processing and other engineering work, you need room. Plotting the results from two data runs with a package like octave is a common task and is difficult without adequate screen room. We have reached the point where using multiple monitors is pretty much the required baseline in order to work.

Fortunately, the rapid obsolescence of computer hardware has resulted in the early retirement of otherwise good equipment. Who doesn't have an extra video card or monitor lying around that was state of the art a few years ago but has been passed by now? Even if you don't, it doesn't cost much to pick up some new hardware and really spice up your system.

We present a formula to set up a Linux system to have two or more displays in various configurations, some lessons we learned while doing this and some uses we've found for the system. Be forewarned—following this formula doesn't guarantee success because some video cards just won't work with others. As always, please do the appropriate backups before making any system changes.

The Formula

To set up your system for multiple cards, start with Red Hat 7.2, two video cards and monitors. We begin with a fresh install so you can see each of the steps in case you've already heavily modified your system. (Follow the spirit of these

steps if you aren't using Red Hat 7.2 or if you are modifying a working installation.) Again, back up your data before trying any of this.

1. Physically install both video cards in the system, turn on both monitors and start the Linux installation.
2. Choose KDE and graphical login when you get to the appropriate part of the installation. This isn't crucial but makes things easier for testing and can be changed later.
3. Finish the installation and make sure your basic system and the X Window System are working by rebooting and logging in. At this time, you'll have one video card and monitor working.
4. As root, launch an xterm and make sure you're in your home directory by typing **cd**. Then type **cp /etc/X11/XFConfig-4 .** to create a backup copy of your X configuration file in your home directory, just in case.
5. Now, type **telinit 3** to switch from runlevel 5, which uses a graphical login, to runlevel 3 with its console login.
6. Time to gather some data and see if this is going to work. Log in again as root. Type **X -version** and note the output. Buried in the output somewhere it will probably say 4.1.0 but could say 3.3.6a, depending on your primary video card. Next is the big step. Type **XFree86 -configure** and wait for the minute or so that this normally takes to run. Now type **XFree86 -xf86config /root/XF86Config.new**. If you are successful, you'll hear clicks from both monitors as they switch modes, and the power lights will switch from amber to green. The screens themselves will fill with the raw X display. If they do, congratulations, you should be able to make this setup work (note: kill X by pressing Ctrl-Alt-Backspace). If they don't, then try a different combination of video cards; we couldn't force systems to work that didn't do this step successfully, but we would love to hear about your successes with it! Now type **XFree86 -scanpci** and record the output. Typical output: **(0:8:0) S3 Virge**, which is a PCI card, and **(1:0:0) NVIDIA Riva Ultra 64**, which is an AGP card. Other cards and slots will be identified, but it doesn't matter for this to work.
7. Configuration of X server: you need to make sure you're calling the right X server, especially if you had an X version less than 4 in step 6. Type **ls -l /etc/X11/X**. If it is a symbolic link to XFree86, you're using the XFree86 Version 4. If it isn't, back up the link by typing **mv /etc/X11/X /etc/X11/X_orig**. Make the correct link by typing **ln -s /etc/X11/bin/XFree86 /etc/X11/X**.
8. Multihead configuration: at this point, you could just copy the new X configuration file (**cp /root/XF86Config.new /etc/X11/XFConfig-4**), but your resolution probably wouldn't be very good. A better approach is to edit /etc/X11/XF86Config-4 and incorporate the knowledge gained from going through these steps. We'll cover the fine-tuning in the next section.

9. Finally, type **telinit 5**, log in and enjoy. If you're successful, your system will boot up and give you a login screen. The screen on the right is your primary screen; the left screen will come up at the end of the login process.

Background

We used Red Hat 7.2 as the baseline because it has XFree86 4, which supports multiple displays (and input devices). Commercial X servers have supported this for a while, but XFree86 4 comes with new Linux distributions and it's free. Version 4 threw in a new configuration file and format to allow this flexibility. The key section is ServerLayout, which positions one screen relative to another. Here's an excerpt:

```
Section "ServerLayout"
        Identifier   "XFree86 Configured"
        Screen   0 "Screen0" 0 0
        Screen   1 "Screen1" LeftOf "Screen0"
        InputDevice  "Mouse0" "CorePointer"
        InputDevice  "Keyboard0" "CoreKeyboard"
EndSection
```

As you undoubtedly realize from this straightforward format, this setup will configure Screen0 (the screen associated with your primary video card) and then configure Screen1 and place it to the left of Screen0. In other words, you would move your mouse across Screen0, hit the left edge and then enter Screen1 from the right side (pretty confusing if you have the monitors backward). You can choose Right, Top, Bottom or even Relative positioning so the screens are diagonal. Do a **man XFree86** to get the details. The screens don't have to be the same resolution or even the same color depth. Make one or both virtual if you want, though we didn't find this easy to use. Each screen in ServerLayout is composed of a Monitor reference and Device (video card) reference as shown below:

```
Section "Screen"
        Identifier "Screen0"
        Device   "Card0"
        Monitor  "Monitor0"
        SubSection "Display"
                Depth   8
                #Virtual         1024 768
        EndSubSection
EndSection
```

Here's the monitor and device sections to support Screen0:

```
Section "Monitor"
        Identifier  "Monitor0"
        VendorName  "GWY"
        ModelName  "311"
EndSection
Section "Device"
        Identifier "Card0"
        Driver   "mga"
        VendorName "Matrox"
        BoardName  "MGA 2164W"
```

```
          BusID    "PCI:0:14:0"
  EndSection
```

The first key thing is to make sure everything is consistent, i.e., Screen0 has Monitor0 and Card0. What's important is that the names match, not what they are. We have developed a habit of making one set end in 0 and the next set end in 1, instead of keeping track of the card and monitor names. It's just easier to avoid mistakes.

The next key thing is the device driver. That gets identified by the **XFree86 -configure** step for both cards. Finally, it's critical to get the BusIDs straight. It's optional with only one card but mandatory for two or more. An AGP card will start with a 1, not a 0, and probably will be 1:0:0.

Now you have the basics and need to fine-tune /etc/X11/XFConfig-4 to get optimal performance. Edit this file and interleave the data identified in /root/XFConfig-4.new. Because we're using KDE's graphical login, you can tweak the file, log out, restart the X server from the KDE login window, do the login and see what happens. If you're not happy, tweak and repeat. It's much faster than manually restarting X. See Listing 1 [available at ftp.linuxjournal.com/pub/lj/listings/issue99/5958.tgz] for a full XFConfig-4 file.

At this point, you have a working multiheaded system utilizing the KDE graphical login and probably the KDE environment (Figure 1). It turns out that there are four different configurations that can be set up: GNOME or KDE login, and then the GNOME or KDE environment. We'll go through the advantages and disadvantages of each approach now.

Figure 1. Successful Multiheaded System in Action

Options

The file /etc/sysconfig/desktop determines which login manager is used. It's a one-line file saying **DESKTOP="GNOME"** or **DESKTOP="KDE"**. Edit as root, according to your requirements. Here are the four configuration options:

1. KDE login manager/KDE work environment (your system is set up this way if you followed the formula). This approach gives you two screens that can be identified by :0.0 for the primary and :0.1 for the secondary. The screens are separate, so you can't move windows from one screen to the other. You have a KDE menubar for each screen. The best use for this is to devote a screen to monitoring (error log, network activity, server room temperature, etc.) whatever you need to keep an eye on that takes up a lot of screen real estate. This is the original motivation for making this work—use some old hardware to monitor network activity and do other work without having another computer on the desk. Since we're working with Linux and have the X Window System, it's easy to control where the X applications appear. You could type **DISPLAY=:0.1 (bash/ksh/sh syntax)**, and then your application will appear in screen :0.1. If you wanted to have just one well-behaved X application in the other screen, type **xclock -display :0.0**, for example.
2. KDE login manager/GNOME work environment (type **switchdesk** from a command prompt and select GNOME to change from the original setup). This gives you a window manager controlling one screen and raw X on the other. You could type **/usr/X11R6/bin/mwm -display :0.1 &** in an xterm (ignore the warning) and get a different window manager running in the other screen. The commands **twm** and **fvwm2** also will work. This is useful for testing purposes because you can quickly see if there are any problems with an application by running it in the environments. As in configuration one, the screens are separate.

Configurations three and four take a different approach and use the Xinerama extensions in XFree86 4. The authors are grateful to Dennis Baker for his splendid HOWTO. Xinerama makes one big screen, so it's probably the easiest and most natural way to work. You just move the windows from one screen to another. There are two key steps for these configurations. First, change the /etc/sysconfig/desktop file as above. Second, in the file /etc/X11/gdm/gdm.conf, edit the line **0=/usr/bin/X11/X** toward the end of the file by making it read **0=/usr/bin/X11/X +xinerama**.

1. GNOME login/GNOME work environment with one large connected screen. When you maximize a window it will fill its screen.
2. GNOME login/KDE work environment. (Select KDE from the login window if you just want to try KDE once.) Again, it's one large connected screen.

This time when you maximize a window it fills the complete screen (spanning between screens).

Lessons

Red Hat has a hardware browser application at /usr/bin/hwbrowser that gives useful information. This is handy under X to see if both video cards are recognized by the system.

Some cards just don't get along. We successfully tried a number of combinations—a matched pair of ATI Mach64 3D RAGE II PCI cards, an ATI Mach64 AGP and Matrox II PCI card, and an S3 Virge PCI and NVIDIA Riva TNT2 AGP card. We got those combinations working within minutes using the above procedure. We also unsuccessfully tried a Diamond Stealth II S220, which definitely does not play well with others. We could not get this card to work with another at all.

The primary video card is where the BIOS display appears and is what gets Screen0. The video card on the highest number PCI channel is the primary card. Some computer BIOSes give an option to initialize the AGP card first, but this did not work for us because the PCI card still displayed the BIOS, which is too bad, because that level of control over the environment would be useful.

Since you have multiple screens when you use the KDE login manager, the KDE screensaver only works on the primary screen. Set up xscreensaver and call it in your shell initialization files (.bashrc for example) if you want both screens to have a screensaver. It's easy to put one screensaver on one screen and a different one on the other using :0.0 or :0.1 in the configuration.

Put some variable assignments in .bashrc, such as **left='-display :0.1'** and **right='-display :0.0'**. Then you can specify which screen to use, e.g., **xeyes $left** or **xclock $right**. It helps make things a little simpler to use.

Conclusion

Our current favorite way to work is to utilize the KDE login and the KDE window environment. KDE seems a little cleaner as a work environment and goes across multiple screens (remember that in configuration two with GNOME, you have to start a second window manager). More importantly, the KDE login gives you separate screens so you can open up one screen for people to use (**xhost +** to disable access control) and still work on the other without security concerns. Also, we tend to devote one screen to monitoring events and displaying things.

The formula presented here is very simple to follow, and a working Linux system can be converted to multiple displays in under ten minutes, if the video

hardware supports it. This technique allowed us to utilize otherwise obsolete hardware to simplify our engineering work greatly.

Acknowledgements

The authors thank Gary Normandin for his insights and hard work testing these methods, Dennis Baker for pointing the way with his Xinerama HOWTO (found at www.linuxdoc.org) and Learning Tree International for letting us use their equipment for some of the testing.

Brian Gollsneider (gollsndb@glue.umd.edu) is working on a PhD in Electrical Engineering from the University of Maryland. When not buried in the clean room, he is a UNIX instructor for Learning Tree International.

Arthur M. Messenger is a retired UNIX system administrator who occasionally answers questions for friends and works part-time for Learning Tree International. When not teaching, he lives with his wife in Haymarket, Virginia and spends time with their grandchildren.

Archive Index Issue Table of Contents

Advanced search

Icarus Verilog: Open-Source Verilog More Than a Year Later

Stephen Williams

Michael Baxter

Issue #99, July 2002

Stephen and Michael provide more technical details on design with Verilog.

About 16 months ago, in the February 2001 *Linux Journal* [see www.linuxjournal.com/article/4428], we reviewed the state of open source in electronic design automation (EDA) with an interview that focused on Stephen Williams, the author of a leading open-source EDA project, the Icarus Verilog compiler [see page 80 of this issue for a new interview with Stephen Williams].

We can look back and forward to the benefits that open-source development methods bring to projects like Icarus Verilog. This article offers more technical details on design with Verilog, as well as explores some of the technical basis for the Icarus Verilog compiler. Also, at the end of this article, we list some excellent reference books on Verilog, as well as web site addresses where you can find information on numerous alternative open-source EDA projects.

Icarus Verilog is a command-line tool that compiles the source design, written in Verilog, to the target format. Normally, the target format is the input to the vvp simulation engine, but there are other target formats that can be selected at the command line.

The obvious first example is the ubiquitous “Hello, World.” program:

```
module test;
initial $display("Hello, World.");
endmodule
```

This compiles and executes trivially with the command sequence:

```
% iverilog -oa.vvp hello.v
% vvp a.vvp
Hello, World
```

The hardware engineer quickly wants to move on to more interesting examples that reflect the problems of hardware design. A simple example that tests a model for a counter is typical (see Listing 1). In both the hello.v and counter1.v examples, the compiler is given a source file that it compiles to a vvp-format output file, and the vvp program executes the generated file.

Listing 1. counter1.v Example

In Verilog programs, modules are the object types that are created by the designer to model a hardware device. Modules instantiate other modules and include code of their own to describe the device being modeled. The designer then instantiates root modules to represent the entire device being modeled.

Verilog compilers typically infer which modules in a design are root modules by noting in the programmer-supplied source which modules are not instantiated anywhere else. In the hello.v example, there was only the module “test”, which was made the root. In the counter1.v program, the module “counter” was instantiated by test, and only test is not otherwise instantiated, so test is the root.

With Icarus Verilog, programmers can either allow this heuristic to guess the root modules or can use -s flags to list root modules explicitly.

As programs get larger, programmers need the ability to create multifile programs and libraries. Libraries are also useful as vendor-supplied models for commercially available devices. Icarus Verilog supports automatic libraries in a portable manner.

An automatic library format is an industry standard. It is a directory of Verilog source files, each containing a module per file with the file named after the module that it contains. For example, counter.v contains the counter module. The location of the library is given to the Icarus Verilog compiler with the -y flag on the command line or in a command file.

Using automatic libraries, our counter1.v example can be broken into two parts: the library modules and the main program. In this example, it makes sense to move the counter module into lib/counter.v and keep the test module in our counter2.v program. See Listing 2 for how to use counter2.v in a two-part manner.

Listing 2. counter2.v Example

Icarus Verilog first tries to compile the counter2.v program. When it gets to the instantiation of the counter module, it notes that it does not have a counter module definition, searches the libraries it knows about (from -y flags) and finds the lib/counter.v file. It parses this new Verilog source, saves the module definitions it finds and continues to compile the original source. Library lookups recurse, so library modules may instantiate other library modules, with the compiler accumulating module definitions until the program is complete.

Icarus Verilog has the further convenience of supporting command files. Command files are text files that contain filenames, path declarations and other compilation directives. We can create for our counter2.v example a command file that looks like this:

```
counter2.txt:
  # this is a private library
  -y lib
  # project source files for this configuration
  counter2.v
```

And run the program like this:

```
% iverilog -c counter2.txt -o a.vvp
% vvp a.vvp
```

For this small program the command file is not much use, but as designs get larger (hundreds of source files is not unusual) the command file becomes indispensable.

Icarus Verilog also includes a unique feature not available to other Verilog compilers: the loadable target API. This is a C API for loadable modules that the compiler can invoke to generate output in new formats. The vvp code generator is itself a loadable target module that is invoked when vvp simulation is requested (the default). There are also the null code generator and the fpga code generator.

The loadable target API is available to C programmers through the ivl_target.h header file included with and installed by Icarus Verilog packages. This API allows C programmers to write new output code generators for the Icarus Verilog compiler.

The vvp runtime also supports a subset of the standard Verilog VPI interface. This is a runtime C API that allows programmers to write new system tasks that the Verilog source can call. Existing examples of system tasks are the $stop and $monitor statements in the counter2.v example source file. All the standard core system tasks are written in Icarus Verilog using the VPI API.

Resources

Stephen Williams got lost one day in the computer lab at his local university. He was an electrical engineering student but emerged with a degree in computer science instead. Ten years later, after working on various software projects, especially large and highly distributed systems, he found himself tangled in the wires of high-speed digital cameras. Since then, he has tied those experiences together, working on device drivers, embedded systems and EDA tools.

Michael Baxter (mabaxter@pacbell.net) has been working in computer technology since he was nine, imprinted by a 1969 viewing of 2001: A Space Odyssey. He is an experienced computer architect, system, board and FPGA logic designer. Michael holds ten US patents in computer architecture and logic, plus five patents as a co-inventor. His interests also include hiking, amateur radio and programming in Lisp.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

A Conversation with Stephen Williams

Michael Baxter

Issue #99, July 2002

Stephen discusses the latest improvements and future plans for Icarus Verilog.

Michael What would you say the strongest improvements to Icarus were in the last year?

Stephen Oh, my—there were so many. I think the most significant improvement has been the simulation engine. By removing the C++ compiler from the Verilog compilation process, I've improved compiler performance several orders of magnitude. I've also made the compiler far more portable. The past year has also seen new compatibility with de facto standards such as library formats and command files. This makes the compiler accessible to more people.

Michael Icarus seems a lot stronger now. In particular, for simulation use Icarus seems to be starting to challenge commercial tools. Could you comment on how you see the quality of results for Icarus so far?

Stephen I never expected to be able to honestly challenge commercial tools, but here I am all the same. Commercial quality is the benchmark I use for judging my progress on Icarus Verilog, and I've made progress in that direction. Most of the feedback I've received in the past has been from people already using commercial tools who notice that Icarus Verilog does something wrong compared to the commercial tool, and that naturally leads to bug fixes that

address limitations compared to the commercial tool.However, with the 0.6 release, the user demographics seem to have shifted a bit. I'm getting more and more feedback from users who do not have other tools. Who knows where this will lead. Comparisons with the big commercial tools are becoming less significant, though.

Michael What improvements for synthesis can we expect from Icarus in the coming year?

Stephen I keep hoping to make real progress on synthesis. However, no one wants to tell anybody how it is done, which leaves me to figure it out for myself. Fortunately, I *have* figured out most of what I need to know to infer logic for almost anything you can express in Verilog. Getting all the technical details worked out will be a lot of work, though.I'm hoping this coming year to get the combinational half of the synthesis all worked out. Quite a lot can be done in this case, and combinational synthesis may be the bigger half. Synchronous logic is a little harder but requires the information needed for combinational synthesis, plus some more.

Michael Today there are many more commercial EDA tools that are supported in the Linux environment. Has this situation affected users of open-source tools like Icarus?

Stephen So far, it appears not to have had a big impact. The set of commercial tools available in the Linux environment is still not complete and varied enough. Having a simulator for Linux is of limited use if your synthesizer and place-and-route tools are still Windows-only. I see that the coming year may bring some positive changes, with some recent announcements of Linux support promising some real change.

Michael IEEE Std 1365-2001, the Verilog 2001 language, still seems pretty new in CAD tools. Have there been issues between the 2001 and 1995 standards that you had to resolve?

Stephen The 2001 standard has resolved a few minor issues of the 1995 standard, but mostly what it has done is standardize, and put into writing, features that many compiler writers were already implementing. Some of the real tricky open issues of the language, including event scheduling, declaration scope and a few others, have not been addressed. The new features of the 2001 standard don't seem to be broken, and I've implemented some of them without any trouble.On the other hand, more complex new features, such as generate statements and configuration files, will likely be slow to be adopted. I know it will take awhile for me to get around to them.

Michael CAD tools often have external utilities to supplement them. You have supported some interesting external formats for open-source graphics tools, like GTKwave. How much work extra work was this?

Stephen Actually, GTKwave is a tool, and the format is VCD (value change dump). In this particular case it was very little work for me because the actual dumper code was submitted by volunteers. Most Verilog programmers consider waveform output fundamentally important; having a standard for the format and off-the-shelf tools to support that format helps very much.This is one of the advantages of standards. A standard that works allows for disparate vendors to produce bits that use the standard to interoperate with other bits. There are so many standards that don't work, we are surprised when we see one that actually accomplishes this goal.

Michael A year ago, compiling speed was an issue. Has the level of quality for the constituent GNU tools since then helped with improving the performance of Icarus?

Stephen No, they have not. The problem was not unique to the GNU tools, though. Compilation with Compaq and Microsoft compilers was at least as slow. The simple truth is that large C++ files as an intermediate form was a really bad idea.

Michael Do you have in mind potentially alternative forms of interoperability in Icarus? For instance, support for Verilog-A or other input formats?

Stephen I have learned to set realistic goals and remain focused. It allows me to make consistent, steady progress for years. The target moves, but it is always just a few steps beyond arm's reach. That means completing coverage of the Verilog standard proper is still the major goal.I have tried to design the interfaces to the core compiler and simulation engine such that related tasks can be done by others, though. As is the UNIX philosophy, I shall endeavor to improve the quality of the interfaces, so I can leave practical extensions to others.

Michael What factors do you feel have most contributed to the success of Icarus?

Stephen I've been lucky enough to have a combination of skills that allowed me to actually understand what I was doing. Once it succeeded past a minimal threshold, Icarus Verilog started to take on a life, and feedback became increasingly more common and useful. Many people seemed to want this, and encouraged me.Over the years, various people put concentrated effort into making patches and giving other forms of feedback. It's hard for folks to keep it

up because no one (in EDA) is paid to work on free software, and free time is scarce and precious, so I managed and dovetailed the feedback the best I could.Surprisingly, I think the high price of commercial EDA tools probably has *not* been terribly significant. Much (but certainly not all) of the effective feedback I received came from people who have and use commercial tools. They know best when I've screwed up. Fortunately for those of us who can't afford those tools, that feedback is forthcoming and we all benefit.

email: mabaxter@pacbell.net

Michael Baxter has been working in computer technology since he was nine, imprinted by a 1969 viewing of 2001: A Space Odyssey. He is an experienced computer architect, system, board and FPGA logic designer. Michael holds ten US patents in computer architecture and logic, plus five patents as a co-inventor. His interests also include hiking, amateur radio and programming in Lisp.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Keeping Up with Python: the 2.2 Release

Wesley J. Chun

Issue #99, July 2002

Python 2.2 resolves some well known deficiencies of the language and introduces some new powerful constructs that are key strengths of other object-oriented languages.

Python 2.2 made its debut at the end of 2001, and its first bug-fix version 2.2.1 was recently released from the core developers at PythonLabs. The 2.2.x family is full of new features and capabilities, some considered significant additions and improvements to the language. These updates give Python developers a significant boost in terms of flexibility.

Python is a simple yet robust language combining the ease of scripting tools with the application-building power of compiled object-oriented programming languages. With Jython, the Java-compiled edition of the Python interpreter, Java programmers are discovering a tool that raises their productivity and development speed to a new level.

You can stay up-to-speed on these changes by reading the PEPs (Python Enhancement Proposals), which are created to give any reasonable idea an ear from the Python community. Before consideration is made for any update to the language, the problems and proposed solutions are presented along with the rationale for, and details behind, the change. Not only can you get the exact details on a PEP at the web site (see Resources), but you can also find out the status of a PEP. After reaching a consensus, a subset of PEPs is approved and slated for each release. For example, the changes in 2.2 (meaning the entire 2.2.x set of releases) consist primarily of five major PEPs: 234, 238, 252, 253 and 255.

For starters, 2.2 begins the process of unifying Python integers and long integers. Integer calculations can no longer raise overflow errors because they will automatically be cast into longs if the value overflows. Statically nested scopes, introduced in 2.1 and now standard, free Python from its restrictive

two-scope model (see PEP 227). Previously, one had to put **from __future__ import nested_scopes** at the start of the script to enable nested scopes. Now that directive is no longer necessary as it has become standard. Unicode support has also been upgraded for UCS-4 (32-bit unsigned integers; see PEP 261). Minor updates to the Python Standard Library include a new e-mail package, a new XML-RPC module, the ability to add IPv6 support to the socket module and the new hot-shot profiler (PEP).

The most significant changes and additions to 2.2 are iterators and generators, changing the division operator and unifying types and classes.

Iterators

Iterators give the programmer the ability to traverse or “iterate through” elements of a data set. They are especially useful when the items of such sets are of differing types. Python has already simplified part of this programming process, as its sequence data types (lists, strings, tuples) are already heterogenous, and iterating through them is as simple as a “for” loop without having to create any special mechanism.

The new iteration support in Python works seamlessly with Python sequences but now also allows programmers to iterate through nonsequence types, including user-defined objects. An additional benefit is the improvement of iteration through other Python types.

Now, that all sounds good, but why iterators in Python? In particular, PEP 234 cites that the enhancement will:

- Provide an extensible iterator interface.
- Bring performance enhancements to list iteration.
- Allow for big performance improvements in dictionary iteration.
- Allow for the creation of a true iteration interface as opposed to overriding methods originally meant for random element access.
- Be backward-compatible with all existing user-defined classes and extension objects that emulate sequences and mappings.
- Result in more concise and readable code that iterates over nonsequence collections (mappings and files for instance).

Iterators can be created directly by using the new iter() built-in function or implicitly for objects that come with their own iteration interface. For example, lists have a built-in iteration interface, so “for eachItem in myList” will not change at all.

Calling iter(obj) returns an iterator for that type of object. An iterator has a single method, next(), that returns the next item in the set. A new exception, StopIteration, signals the end of the set.

Iterators do have restrictions, however. You can't move backward, go back to the beginning or copy an iterator. If you want to iterate over the same objects again (or simultaneously), you have to create another iterator object.

Sequences

As mentioned before, iterating through Python sequence types is as expected:

```
>>> myTuple = (123, 'xyz', 45.67)
>>> i = iter(myTuple)
>>> i.next()
123
>>> i.next()
'xyz'
>>> i.next()
45.67
>>> i.next()
Traceback (most recent call last):
  File "", line 1, in ?
StopIteration
```

If this had been an actual program, we would have enclosed the code inside a try-except block. Sequences now automatically produce their own iterators, so a “for” loop:

```
for i in seq:
    do_something_to(i)
```

under the covers now really behaves like this:

```
fetch = iter(seq)
while 1:
    try:
        i = fetch.next()
    except StopIteration:
        break
    do_something_to(i)
```

However, your code doesn't need to change because the “for” loop itself calls the iterator's next() method.

There is also another form of the iter() built-in function, iter(callable, sentinel), which returns an iterator as before. The difference is that each call to the iterator's next() method will invoke callable() to obtain successive values and raise StopIteration when the value sentinel is returned.

Dictionaries

Dictionaries and files are two other Python data types that received the iteration makeover. A dictionary's iterator traverses its keys. The idiom “for

eachKey in myDict.keys()" can be shortened to "for eachKey in myDict", as shown in Listing 1.

Listing 1. Looping through a Dictionary

In addition, three new built-in dictionary methods have been introduced to define the iteration: myDict.iterkeys() (iterate through the keys), myDict.itervalues() (iterate through the values) and myDict.iteritems() (iterate through key/value pairs). Note that the "in" operator has been modified to check a dictionary's keys. This means the Boolean expression myDict.has_key(anyKey) can be simplified as "anyKey in myDict".

Files

File objects produce an iterator that calls the readline() method. Thus, they loop through all lines of a text file, allowing the programmer to replace essentially "for eachLine in myFile.readlines()" with the more simplistic "or eachLine in myFile":

```
>>> myFile = open('config-win.txt')
>>> for eachLine in myFile:
...     print eachLine,   # comma suppresses extra \n
...
[EditorWindow]
font-name: courier new
font-size: 10
>>> myFile.close()
```

Classes

You can also create custom iterators for your own classes. This allows you to avoid the hack of overloading the __getitem__() special class method. Overloading __getitem__() implies the user can ask for any subscript in any order. But some objects do not logically allow this. Using an iterator rather than overloading __getitem__() makes explicit what the user can or cannot do.

To add iteration to your classes, override the __iter__() special method to return itself (making the object its own iterator). Then override the next() method:

```
def __iter__(self):
    return self
def next(self):
    # return next item or raise StopIteration
```

We can tweak our code for a similar example. This time, we choose to return a random element from the sequence (Listing 2). This example demonstrates some unusual things we can do with custom class iterations. One is infinite iteration. Because we read the sequence nondestructively, we never run out of elements, so we never need to raise StopIteration.

Listing 2. Custom Class Iterations

In Listing 3, we create an iterator object using our class, but rather than iterating through one item at a time, we give the next() method an argument telling how many items to return.

Listing 3. Creating an Iterator Object Using Our Class

Now let's try it out:

```
>>> a = AnyIter(range(10))
>>> i = iter(a)
>>> for j in range(1,5):
>>>     print j, ':', i.next(j)
1 : [0]
2 : [1, 2]
3 : [3, 4, 5]
4 : [6, 7, 8, 9]
```

Mutable Objects and Iterators

Before we move on to generators, remember that interfering with mutable objects while you are iterating them is not a good idea. This was a problem before iterators appeared. One popular example of this is to loop through a list and remove items from it if certain criteria are met (or not):

```
for eachURL in allURLs:
    if not eachURL.startswith('http://'):
        allURLs.remove(eachURL)            # YIKES!!
```

All sequences are immutable except lists, so the danger occurs only there. A sequence's iterator only keeps track of the *N*th element you are on, so if you change elements around during iteration, those updates will be reflected as you traverse through the items. If you run out, then StopIteration will be raised, but you can continue with the iteration if you add items to the end and resume, as shown in Listing 4.

Listing 4. Iteration Example

When iterating through keys of a dictionary, you must not modify the dictionary. Using a dictionary's keys() method is okay because keys() returns a list that is independent of the dictionary.

But iterators are tied much more intimately with the actual object and will not let us play that game anymore:

```
>>> myDict = {'a': 1, 'b': 2, 'c': 3, 'd': 4}
>>> for eachKey in myDict:
...   print eachKey, myDict[eachKey]
...   del myDict[eachKey]
...
a 1
```

```
Traceback (most recent call last):
  File "", line 1, in ?
RuntimeError: dictionary changed size during
              iteration
```

This will help prevent buggy code. For full details on iterators, see PEP 234.

Generators

Generators extend from the idea of iterators. The main motivation for generators, however, comes from a different angle: they allow saving state across function calls. Static variables, such as in C functions, have the ability to maintain their value across multiple calls of that function. This partially solves the state problem, but what would be really nice would be to yield a value just like an iterator but be able to freeze execution only to resume exactly where you left off when it is called again. This is exactly what generators do. They represent the idea of merging iteration with state along with functions that are resumable. When they do, they pick up right where they left off, keeping intact all the state information they need to deliver the next item. Note that we use the term yield here for two reasons: to hint that it's not a true return (along with frame object stack pop) and to introduce the new keyword yield.

For backward compatibility (in case there is code out there that uses yield as an identifier), you must include the “from __future__ import generators” directive to use generators. Generators will become standard soon (2.3?) so importing will not be a necessity. Generators behave in another manner similar to iterators: when a real return or end-of-function is reached and there are no more values to yield, a StopIteration exception is raised. Here's a simple example:

```
def simpleGen():
    yield 1
    yield '2 --> punch!'
```

Now that we have our function, let's call it and get a generator object:

```
>>> myG = simpleGen()
>>> myG.next()
1
>>> myG.next()
'2 --> punch!'
>>> myG.next()
Traceback (most recent call last):
  File "", line 1, in ?
    myG.next()
StopIteration
```

or more aptly: for eachItem in simpleGen(): print eachItem. Of course that was a silly example. I mean, why not use a real iterator for that? More motivation comes from being able to iterate through a sequence that requires the power of a function, rather than static objects already sitting in some sequence.

In the following example, we are going to create a random iterator that takes a sequence and returns a random item from that sequence:

```
from random import randint
def randIter(seq):
    while len(seq) > 0:
        yield seq.pop(randint(0, len(seq)-1))
```

The difference is that each item returned is also consumed from that sequence, sort of like a combination of list.pop() and random.choice():

```
>>> for eachItem in randIter([123, 'xyz',
    45.678, 9j]):
...     print eachItem
...
'xyz'
9j
45.678
123
```

Table 1 is a summary of the differences between iterators and generators. You can find more details on both iterators and generators in their respective PEPs (234 and 255).

Table 1. Differences between Iterators and Generators

Initiating the Process of Changing the Division Operator

This is perhaps the most controversial update to Python so far. There are many pros and cons, but finally those who believe in true division have won out. To highlight this change let's define (or redefine) some division terms and their functionality with integer and floating-point operands.

Classic Division

When presented with integer operands, classic division truncates the decimal place, returning an integer (see the “Floor Division” section below). When given a pair of floating-point operands, it returns the actual floating-point quotient (see the “True Division” section). Here is an example of what Python's division has been and still is today (actually a mix of true and floor division):

```
>>> 1 / 2          # perform integer result (floor)
0
>>> 1.0 / 2.0      # returns real quotient
0.5
```

True Division

This is the case where the result should always be the actual quotient, regardless of the type of the operands. This is the big change that is to come our way when Python 3.0 nears reality. For now, to take advantage of true

division, one must give the **from __future__ import division** directive. Once that happens, the division operator (/) performs only true division:

```
>>> from __future__ import division
>>>
>>> 1 / 2               # returns real quotient
0.5
>>> 1.0 / 2.0           # returns real quotient
0.5
```

Floor Division

A new divisor operator (//) has been created that always truncates the fraction and rounds it to the next smallest whole number toward the left on the number line, regardless of the operands' numeric types. This operator works starting in 2.2 and does not require the __future__ directive above.

```
>>> 1 // 2          # floors result, returns integer
0
>>> 1.0 // 2.0      # floors result, returns float
0.0
>>> -1 // 2         # move left on number line
-1
```

Without getting into the arguments of this change, the feeling is that perhaps Python's division operator has been flawed since the beginning, especially because Python is a strong choice as a first programming language for people who aren't used to floor division. One of the examples Guido uses in his “What's New in Python 2.2” ZPUG talk is:

```
def velocity(distance, totalTime):
    rate = distance / totalTime
```

This is bad because this function is not numeric-type-independent. Your results with a pair of floats certainly differs from that of sending in a pair of integers. To bridge the dichotomy, you must resolve the following intransitivity in your head:

```
>>> 1 == 1.0
1
>>> 2 == 2.0
1
>>> 1 / 2 == 1.0 / 2.0           # classic division
0
```

If you use Python's new model of division, the universe is at peace once again:

```
>>> from __future__ import division
>>> 1 / 2 == 1.0 / 2.0           # true division
1
>>> 1 // 2 == 1.0 // 2.0         # floor division
1
```

While this seems like the proper and right thing to do, one cannot help but be concerned with the code breakage it may lead to. Fortunately, the Python developers have kept this in mind, as this change will not be permanent until Python 3.0, which is still years away. Those who desire the new division can

import it or start Python with the -Qnew command-line option. There are a few options to turn on warnings to prepare for the upcoming new division.

You can get more information from PEP 238, but dig through the comp.lang.python archives for the heated debates. Table 2 summarizes the division operators in the various releases of Python and the differences in operation when you import division (from __future__).

Table 2. Division Operator Summary

Merging Types and Classes

Merging Python types and classes has been on the want list for quite a while. Programmers are dismayed to discover that they cannot subclass existing data types, such as a list, to customize for their applications.

To learn more, it can't hurt to look through both the PEPs involved and a tutorial Guido wrote specifically for those who want to get up to speed quickly on the new style classes without having to wade through all the intricate details found in the PEPs (see Resources). We will also give you a teaser class that extends a Python list with enhanced stack features.

This example, stack2.py, is motivated by one of the iterator examples above (see also Example 6.2 at the *Core Python Programming* web site).

```
#!/bin/env python
'stack2.py -- subclasses and extends a list'
class Stack(list):
  def __init__(self, *args):
      list.__init__(self, args)   # call base class
                                  # constructor
  def push(self, *args):
      for eachItem in args:       # can push multiple
          self.append(eachItem)   # items
  def pop(self, n=1):
      if n == 1:                  # pop single item
          return list.pop(self)
      else:                       # pop multiple items
          return [ list.pop(self) for i in range(n) ]
```

Below is the output we get from flexing our newfound capabilities:

```
>>> from stack2 import Stack
>>> m = Stack(123, 'xyz')
>>> m
[123, 'xyz']
>>> m.push(4.5)
>>> m
[123, 'xyz', 4.5]
>>> m.push(1+2j, 'abc')
>>> m
[123, 'xyz', 4.5, (1+2j), 'abc']
>>> m.pop()
'abc'
>>> m.pop(3)
[(1+2j), 4.5, 'xyz']
```

```
>>> m
[123]
```

In addition to being able to subclass built-in types, other highlights of the new style classes include:

- “Cast” functions being factories.
- New __class__, __dict__, and __bases__ attributes.
- __getattribute__() Special Method (smarter than __getattr__()).
- Class descriptors.
- Class properties.
- Static methods.
- Class methods.
- Superclass method calls.
- Cooperative methods.
- New diamond diagram name resolution.
- Fixed set of allowed class attributes with Slots.

For more information on the new style classes and the unification of types and classes, see both PEPs 252 and 253 as well as the aforementioned tutorial by Guido.

Conclusion

Although all these new features and weakness resolutions bring Python pretty far down the path, there are those who claim that they violate Python's simplistic nature. If you're stictly a purist, that is probably a valid consideration. However, by finally purging some of the annoyances and adding a few more powerful constructs to the language, we are probably better off than we were before. These changes will not have a negative impact on existing code, and those that do, such as the change in the division operator, are at least not required for some time and allow for a more painless transition.

Finally, see Resources for a couple of other high-level documents, such as Andrew Kuchlin's “What's New in Python 2.2” and the slide presentation from one of Guido's talks last fall at a Python user group meeting. Python 2.2.1 can be downloaded at the main Python language home page. Happy hacking!

Resources

email: cyberweb@rocketmail.com

Wesley J. Chun, author of Core Python Programming, has over a decade of programming and instructional experience. Chun helped build Yahoo! Mail and Yahoo! People Search using Python and is currently employed by Synarc, a service company in clinical trials utilizing Python to develop applications that allow radiologists to perform patient assessments.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

DSI: Secure Carrier-Class Linux

The DSI Team

Issue #99, July 2002

Ericsson Research is developing a new secure architecture for telecom applications on Linux clusters.

The interest in clustering from the telecommunications industry originates with the fact that clusters address carrier-class characteristics, such as guaranteed service availability, reliability and scaled performance, using cost-effective hardware and software. These carrier-class requirements now include advanced levels of security. There are few efforts, however, to build a coherent distributed framework to provide advanced security levels in clustered systems.

At Ericsson Research, our work targets soft real-time distributed applications running on large-scale Linux carrier-class clusters. These clusters must operate nonstop and must allow operators to upgrade hardware and software during operation, without disturbing the applications that run on them. In such clusters, communications between the nodes inside the cluster and to external computers are restricted.

In this article, we present the rationale behind developing a new secure architecture, the DSI (Distributed Security Infrastructure). DSI supports different security mechanisms to address the needs of telecom applications running on carrier-class Linux clusters. DSI provides these telecom applications with distributed mechanisms for access control, authentication, auditing and integrity of communications.

The Need for a New Security Approach

Many security solutions exist for clustered servers, but no solution is dedicated to clusters.

The most commonly used security approach is to package several existing solutions. Nevertheless, the integration and management of these different

packages is complex and often results in the absence of interoperability between different security mechanisms. Additional difficulties also are raised when integrating many packages, issues like ease of system maintenance and upgrade, and difficulty keeping up with numerous security patches.

Carrier-class clusters have very tight restrictions on performance and response time, making the design of security solutions difficult. In fact, many security solutions cannot be used due to their high-resource consumption.

Currently implemented security mechanisms are based on user privileges and do not support authentication and authorization checks for interactions between two processes belonging to the same system on different processors. However, for telecom applications, only a few users run the same application for a long period without any interruption.

Applying the above concept will grant the same security privileges to all processes created on different nodes. This would lead to no security checks for many actions through the distributed system.

DSI Characteristics

As part of a carrier-class Linux cluster, DSI must comply with the carrier-class requirements of reliability, scalability and high availability. Furthermore, DSI supports the following requirements: 1) Coherent framework: security must be coherent across different layers of heterogeneous hardware, applications, middleware, operating systems and networking technologies. All mechanisms must fit together to prevent any exploitable security gap in the system. 2) Process-level approach: DSI is based on a fine-grained basic entity, the process. 3) Minimal performance impact: the introduction of security features must not impose high-performance penalties. Performance can be expected to degrade slightly during the initial establishment of a security context; however, the impact on subsequent accesses must be negligible. 4) Preemptive security: changes in the security context will be reflected immediately on the running security services. Whenever the security context of a subject changes, the system will re-evaluate its current use of resources against this new security context. 5) Dynamic security policy: it must be possible to support runtime changes in the distributed security policy. Carrier-class server nodes must provide continuous and long-term availability; thus, it is impossible to interrupt the service to enforce a new security policy. 6) Transparent key management: cryptographic keys are generated in order to secure connections. This results in numerous keys that must be stored and managed securely.

DSI Architecture

DSI has two types of components: management and service. DSI management components define a thin layer that includes a security server, security managers and a security communication channel (Figure 1). The service components define a flexible layer that can be modified or updated by adding, replacing or removing services according to the needs.

The security server is the central point of management in DSI, the entry point for secure operation and management and intrusion detection systems. It also defines the dynamic security environment of the whole cluster by broadcasting changes in the distributed policy to all security managers.

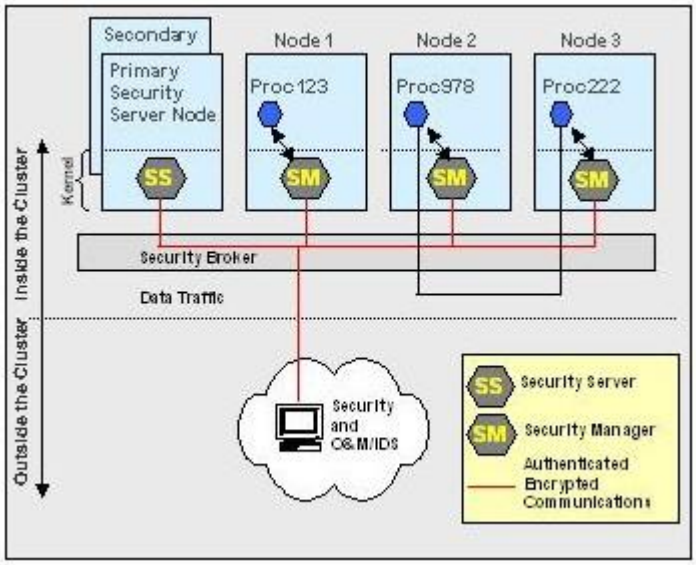


Figure 1. Distributed Architecture of DSI

Security managers enforce security at each node of the cluster. They are responsible for locally enforcing changes in the security environment. Security managers only exchange security information with the security server.

The secure communication channel provides encrypted and authenticated communications between security agents. All communications between the security server and the world outside of the cluster take place through the secure communication channel. Two nodes (to avoid a single point of failure) host the security server and different security service providers, such as the certification authority.

The security mechanisms are based on widely known, proved and tested algorithms. Users must not be able to bypass these mechanisms; therefore, the best place to enforce security is at the kernel level. All security decisions, when necessary, are implemented at the kernel level, the same as for the main security manager component, which has stubs into the kernel. These stubs are implemented through load modules.

The DSI architecture at each node is based on a set of loosely coupled services. Each service, upon its creation, sends a presence announcement to the local security manager, which registers these services and provides their access mechanisms to the internal modules. Two types of services, security services (access control, authentication, integration, auditing) and security service providers (for example, secure key management), run at user level and provide services to security managers.

Current Results

So far, a secure boot mechanism for diskless Linux servers has been implemented. Using secure boot with digital signatures, a distributed trusted computing base (DTCB) will be available at the boot of the cluster nodes. The kernel at secure boot is small enough to be thoroughly tested for vulnerabilities. Furthermore, the use of digital signatures for binaries and a local certification authority will prevent malicious modifications to the DTCB.

We also implemented a security module based on the Linux Security Module (LSM), which enforces the security policy as part of the DSI access control service. This module is integrated with SCC to implement distributed access control mechanisms. DSI currently supports preemptive and dynamic security policy at the process level throughout the whole cluster.

To ease administration and maintenance of the distributed security policy, we are completing a study to devise methods of reusing information already contained in package management systems (such as RPM) in order to generate part of the security policy or to push such information to the package (if that is where it would be best specified). This effort also aims to use the policy to provide clearly different privileges during software installation, configuration, activation and execution. Specification of the exact language used to express the policy and of the compilation and loading mechanisms remain to be completed.

We have partly implemented a secure communication channel based on OmniORB, an open-source implementation of CORBA. SCC logics are implemented on top of a portability layer, making the implementation independent of any communication middleware used. The choice of CORBA as communication middleware for SCC was motivated by many factors, such as the support for distributed real-time and embedded systems and interoperability.

DSI as an Open-Source Project

Our goal for DSI is to make the framework open source and to get people from different organizations and open-source initiatives involved in the design and development of the various components.

Figure 2 presents the various components of DSI. All components with a question mark are open to design and development contribution. Currently at Ericsson Research we are working toward implementing the core DSI, which includes the following: secure communication channel, security server, security manager, access control service (including LSM), security policy generation, security session manager and distributed tracing of events (as part of the auditing service).

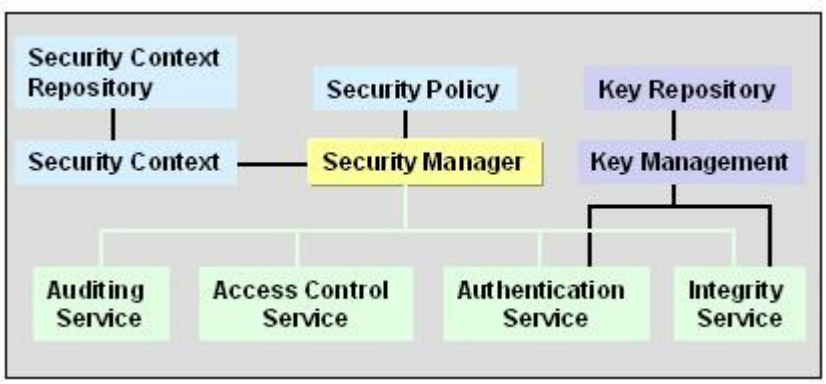


Figure 2. DSI Components

Ericsson Joins the Open Source Development Lab

Conclusion

The DSI team from Ericsson Research will be available at the Ottawa Linux Symposium for three allocated presentations on DSI. We will also be available at the IEEE Cluster Conference 2002 in Chicago. In addition, Ericsson Research will be hosting the annual Open Cluster Group meetings June 24-25, in Montréal, which will give us the opportunity to address the members of the group and get them involved with the DSI Project.

A web site for the project is available as of June 2002. It provides DSI technical reports, presentations, source code and links to web sites of other contributors. Due to space limitations, we were not able to go into the details of DSI in this article. However, feel free to contact any of the DSI team members (listed below) to receive detailed papers on the DSI architecture, strategy and source code or to discuss collaboration opportunities.

The DSI Team

Ibrahim F. Haddad (Ibrahim.Haddad@Ericsson.com), Charles Levert (Charles.Levert@Ericsson.ca), Makan Pourzandi (Makan.Pourzandi@Ericsson.ca) and Miroslaw Zakrzewski (Miroslaw.Zakrzewski@Ericsson.ca).

Contributors

Marc Chatel (Marc.Chatel@lmc.Ericsson.se), Michel R. Dagenais (Michel.Dagenais@polymtl.ca), David Gordon (David.Gordon@Ericsson.ca), Bruno J. M. Hivert (Bruno.Hivert@Ericsson.com) and Dominic Pellerin (Dominic.Pellerin@Ericsson.ca).

email: ibrahim.haddad@lmc.ericsson.se

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Proper Linux Kernel Coding Style

Greg Kroah-Hartman

Issue #99, July 2002

Some simple rules, written and unwritten, to ensure that other kernel users understand your work.

As more people and companies start to write Linux kernel code, understanding acceptable kernel programming coding style and conventions is very important. This article starts out by explaining the written kernel coding style, as documented in the kernel's Documentation/CodingStyle file, and then moves on to introduce some of the “unwritten” kernel coding style rules.

Why Rules?

Why are there kernel programming style rules in the first place? Why not just let authors code in whatever style they want, and let everyone live with it? After all, code formatting does not affect memory use, execution speed or anything else a normal user of the kernel would see. The reason for rules can be summed up with this passage from Elliot Soloway and Kate Ehrlich's article (see Resources):

> Our studies support the claim that knowledge of programming plans and rules of programming discourse can have a significant impact on program comprehension. In their book called *Elements of Style*, Kernighan and Plauger also identify what we would call discourse rules. Our empirical results put teeth into these rules: it is not merely a matter of aesthetics that programs should be written in a particular style. Rather there is a psychological basis for writing programs in a conventional manner: programmers have strong expectations that other programmers will follow these discourse rules. If the rules are violated, then the utility afforded by the expectations that programmers have built up over time is effectively nullified.

There have been a number of other studies about how a large body of code written in a common style directly affects how easy it is to understand the code, review it and revise it quickly.

Because the number of developers that look at the Linux kernel code is very large, it is in the best interest for the project to have a consistent style guideline. This allows the code to be easier to understand by both someone reading it for the first time and for someone revisiting his or her old code a number of months later. It also allows someone else to read easily, understand and potentially fix and enhance your code, which is one of the greatest strengths of open-source code.

What Are the Rules?

So now that we have an understanding of why there should be rules, what are they? Linus Torvalds and other kernel programmers have written a short document that details some of the kernel programming rules. This document is located in the Documentation/CodingStyle file in the kernel source tree. It is required reading for anyone who wants to contribute to the Linux kernel. The following is a summary of these rules.

Indentation

All tabs are eight characters and will be the <TAB> character. This makes locating where different blocks of code start and end easier. If you find that your code is indented too deeply, with more than three levels of indentation that cause the code to shift to the right of the screen too far, then you should fix this.

Placing Braces

The original authors of UNIX placed their braces with the opening brace last on the line, and the closing brace first on the line:

```
if (x is true) {
        we do y
}
```

Therefore, the Linux kernel uses this style. The exceptions to this rule are functions, which have the opening brace at the beginning of the line:

```
int function(int x)
{
        body of function
}
```

Again, this is how Kernighan and Ritchie wrote their code.

For good examples of proper indentation and brace style, look at any of the fs/*.c files or anything in the kernel/*.c files. Generally, most of the kernel is in the proper indentation and brace style, but there are some notable exceptions. The code in fs/devfs/*.c and drivers/scsi/qla1280.* are good examples of how *not* to do indentation and braces.

A script can be used to run the indent program if you have to convert a large amount of code to the correct format. This file is located at scripts/Lindent in the kernel source tree.

Naming

Your variables and functions should be descriptive and concise. Don't use long flowery names like, CommandAllocationGroupSize or DAC960_V1_EnableMemoryMailboxInterface() but rather, cmd_group_size or enable_mem_mailbox(). Mixed-case names are frowned upon, and encoding the type of the variable or function in the name (like “Hungarian notation”) is forbidden.

Global variables should be used only if they are absolutely necessary. Local variables should be short and to the point. Valid local loop variable names include “i” and “j”, while “loop_counter” would be too verbose; “tmp” is allowed for any short-lived temporary variable.

Again, good examples of proper names can be found in fs/*.c. A lot of driver code has bad variable names, as it has been ported from other operating systems. Examples of how *not* to name functions and variables include drivers/block/DAC960.* and drivers/scsi/cpqfs*.

Functions

Functions should do only one thing and do it well. They should be short and hopefully contain only one or two screens of text. If you have a function that does a lot of small things for different cases, it is acceptable to have a longer function. But if you have a long, complex function, you have a problem.

In addition, a large number of local variables within a function is a measure of how complex the function is. If the number of local variables exceeds ten, then there is something wrong.

Again, there are a lot of good examples of nice-sized functions in the fs/*.c and other kernel core code. Some bad examples of functions can be found in drivers/hotplug/ibmphp_res.c (where one function is 370 lines long) or drivers/usb/usb-uhci.c (where one function has 18 local variables).

Comments

Comments are good to have, but they have to be useful. Bad comments explain how the code works, who wrote a specific function on a specific date or other such useless things. Good comments explain what the file or function does and why it does it. They should also be at the beginning of the function and not necessarily embedded within the function (you are writing small functions, right?).

Now there is also a standard format for function comments. It is a variant of the documentation method used by the GNOME Project for its code. If you write your function comments in this style, the information in them can be extracted by a tool and made into standalone documentation. This can be seen by running **make psdocs** or **make htmldocs** on the kernel tree to generate a kernel-api.ps or kernel-api.html file containing all of the public interfaces to the different kernel subsystems.

This style is documented in the file Documentation/kernel-doc-nano-HOWTO.txt and scripts/kernel-doc. The basic format is as follows:

```
/**
 * function_name(:)? (-
(* @parameterx: (
(* a blank line)?
 * (Description:)? (
 * (section header: (
(*)?*/
```

The short function description cannot run multiple lines, but the other descriptions can (and they can contain blank lines). All further descriptive text can contain the following markups:

- funcname(): function
- $ENVVAR: environment variable
- &struct_name: name of a structure (up to two words, including struct)
- @parameter: name of a parameter
- %CONST: name of a constant

So a simple example of a function comment with a single argument would be:

```
/**
 * my_function - does my stuff
 * @my_arg: my argument
 *
 * Does my stuff explained.
 **/
```

Comments can and should be written for structures, unions and enums. The format for them is much like the function format:

```
/**
 * struct my_struct - short description
 * @a: first member
 * @b: second member
 *
 * Longer description
 */
struct my_struct {
    int a;
    int b;
};
```

Some good examples of well-commented functions can be found in the drivers/usb/usb.c file, where all global functions are documented. The file arch/i386/kernel/mtrr.c is a good example of a file with a reasonable amount of comments, but they are in the incorrect format, so they cannot be extracted by the documentation tools. A good example of how *not* to create the comment blocks for your functions is drivers/scsi/pci220i.c.

Data Structure Requirements

A chapter on data structures showed up in the 2.4.10-pre7 kernel. It goes into how every data structure that can exist outside of a single-threaded environment needs to implement reference counting in order to handle the memory management issues of the structure properly. If you add reference counting to your structure, you can avoid a lot of nasty locking issues and race conditions. Multiple threads can access the same structure without having to worry that a different thread will free the data from under it.

The last sentence in this chapter is required reading by any kernel developer: “Remember: if another thread can find your data structure, and you don't have a reference count on it, you almost certainly have a bug.”

A good example of why reference counting is necessary can be found in the USB data structure, “struct urb”. This structure is created by a USB device driver, filled with data, sent to a USB host controller where it will be processed and eventually sent out over the wire. When the host controller is finished with the urb, the original device driver will be notified. While a host controller driver is processing the urb, the original driver can try to cancel the urb or even free it. This led to numerous bugs in the core USB subsystem and different drivers. It also created long, detailed arguments on the linux-usb-devel mailing list about when in the life span of a urb it was allowed to be touched by either driver. In the 2.5 kernel series, struct urb had a reference count added to it, and the USB core and USB host controller drivers had a small amount of code added to handle the reference count properly. Now, whenever a driver wants to use the urb, a reference count is incremented. When it is finished with it, the reference count is decremented. If this was the last user, the memory is then freed and the urb disappears. This allows the USB device drivers to simplify their urb-

handling logic vastly, fixes a lot of different race condition bugs and quiets all arguments about the topic.

Unwritten Rules

If you follow the above set of rules, your code will look like good Linux kernel code. However, quite a few unwritten rules and style guidelines exist that good kernel code follows. Here are some of them.

Avoid NIH Syndrome

A wide variety of well-designed, well-documented and well-debugged function and data structures can be found within the kernel. Please take advantage of them, and do not reinvent your own version of them just because you did not write them. Among the most common of these are the string functions, the byte order functions and the linked list data structure and functions.

String Functions

In the file include/linux/string.h, a number of common string handling functions are defined. These include strpbrk, strtok, strsep, strspn, strcpy, strncpy, strcat, strncat, strcmp, strncmp, strnicmp, strchr, strrchr, strstr, strlen, strnlen, memset, memcpy, memove, memscan, memcmp and memchr. And a number of “simple” string functions are defined in the file, include/linux/kernel.h: simple_strtoul, simple_strtol, simple_strtoull and simple_strtoll. So, if you need any type of string functionality in your kernel code, please use the built-in functions. Do not try to rewrite the existing functions accidentally.

Byte Order Handling

Do not rewrite code to switch data between different endian representations. The file include/asm/byteorder.h (asm will point to the proper subdirectory, depending on your processor architecture) will bring in a wide range of functions that allow you to do automatic conversions, regardless of the endian format of your processor or your data.

Linked Lists

If you need to create a linked list of any kind of data structure, use the code that is in include/linux/list.h. It contains a structure, struct list_head, that should be included within the structure of which you want to create a list. Then you will be able to add, remove or iterate over a list of data structures easily, without having to write new code.

Some good examples of code that uses the list structure can be found in drivers/hotplug/pci_hotplug_core.c and drivers/ieee1394/nodemgr.c. Some

code in the kernel that should be using the list structure can be found in the ATM core, within the struct atm_vcc data structure. Because the ATM code did not use struct list_head, every ATM driver needs to walk the lists of data structures by hand, duplicating a lot of code.

typedef Is Evil

typedef should not be used in naming any of your structures. Most main kernel structures have no typedef. Though this would shorten their usage, it obscures code. For example, the core kernel structures struct inode, struct dentry, struct file, struct buffer_head, struct user and struct task_struct have no typedef.

Using a typedef only hides the real type of a variable. There are records of some kernel code using typedefs nested up to four layers deep, preventing the programmer from easily telling what type of variable they are actually using. If the programmer does not realize the size of the structure it can cause very large structures to be accidentally declared on the stack or to be returned from functions.

typedef also can be used as a crutch to keep from typing long structure definitions. If this is the case, the structure names should be made shorter, according to the above-listed naming rules.

Never define a typedef to signify a pointer to a structure, as in the following example:

```
typedef struct foo {
        int bar;
        int baz;
} foo_t, *pfoo_t;
```

This hides the true type of the variable and uses the name of the variable type to define what it is (see the previous comment about Hungarian notation).

Some examples of where typedefs are used badly are in the include/raid/md*.h files, where every structure has a typedef assigned to it, and in the drivers/acpi/include/*.h files, where a lot of the structures do not even have a name assigned to them, only a typedef.

The only place that using typedef is acceptable is in declaring function prototypes. These can be difficult to type out every time, so declaring a typedef for these is nice to do, for example, the bh_end_io_t typedef that is used as a parameter in the init_buffer() call. This is defined in include/fs.h as:

```
typedef void (bh_end_io_t)
    (struct buffer_head *bh, int uptodate);
```

No ifdefs in .c Code

With the wide number of different processors, different configuration options and variations of the same base hardware types that Linux runs on, it is easy to start having a lot of #ifdef statements in your code. This is not the proper thing to do. Instead, place the #ifdef in a header file, and provide empty inline functions if the code is not to be included.

As an example, consider the following code in drivers/usb/hid-core.c:

```
static void hid_process_event (struct hid_device *hid,
                               struct hid_field *field,
                               struct hid_usage *usage,
                               __s32 value)
{
   hid_dump_input(usage, value);
   if (hid->claimed & HID_CLAIMED_INPUT)
         hidinput_hid_event(hid, field, usage, value);
#ifdef CONFIG_USB_HIDDEV
   if (hid->claimed & HID_CLAIMED_HIDDEV)
         hiddev_hid_event(hid, usage->hid, value);
#endif
}
```

Here the author does not want to call hiddev_hid_event() if a specific configuration option is not enabled. This is because that function will not even be present if the configuration option is not enabled.

To remove this #ifdef, the following change was made to include/linux/hiddev.h:

```
#ifdef CONFIG_USB_HIDDEV
   extern void hiddev_hid_event (struct hid_device *,
                                 unsigned int usage,
                                 int value);
#else
   static inline void
   hiddev_hid_event (struct hid_device
*hid,
                     unsigned int usage,
                     int value) { }
#endif
```

Then drivers/usb/hid-core.c was changed to:

```
static void hid_process_event
                           (struct hid_device *hid,
                            struct hid_field *field,
                            struct hid_usage *usage,
                            __s32 value)
{
   hid_dump_input(usage, value);
   if (hid->claimed & HID_CLAIMED_INPUT)
         hidinput_hid_event(hid, field, usage, value);
   if (hid->claimed & HID_CLAIMED_HIDDEV)
         hiddev_hid_event(hid, usage->hid, value);
}
```

If CONFIG_USB_HIDDEV is not enabled, the compiler will replace the call to hiddev_hid_event() with a null function call and then optimize away the if

statement entirely. This keeps the code readable and easier to maintain over time.

Conclusion

The Linux kernel consists of a large amount of source code, contributed by hundreds of developers over many years. Because the majority of this code follows some simple and basic style and formatting rules, the ability for people to understand new code quickly has been greatly enhanced. If you want to contribute to this code, please read the CodingStyle guidelines and follow them in your patches and new code. The other unwritten rules can be just as important as the written ones when you are trying to convince people to accept your contributions, and they should be followed just as closely.

This article was based upon a paper and presentation that was created for the 2002 Ottawa Linux Symposium.

Resources

Greg Kroah-Hartman is currently the Linux USB and PCI Hot Plug kernel maintainer. He works for IBM doing various Linux kernel-related things and can be reached at greg@kroah.com.

LINUX JOURNAL

Apache 2.0

Reuven M. Lerner

Issue #99, July 2002

Reuven discusses the significance of the 2.0 release for web developers, administrators and the Open Source community.

As I write this, Apache 2.0 has been out in stable form for nearly a month—and from everything I can tell, it's definitely ready for prime time. While there are other open-source HTTP servers, Apache is definitely the best known and best supported. Apache is used on 60% of the web sites in the world, comes with virtually every Linux distribution and is even part of several commercial application servers. Both Zope and Jakarta-Tomcat have their own built-in HTTP servers, but almost no one exposes these servers directly to the Web. Rather, they use Apache as a front end because of its combination of performance and flexibility. This month, we take a closer look at Apache 2.0 [see also “Apache 2.0: the Internals of the New, Improved 'A PatCHy'", available at www.linuxjournal.com/article/4559].

Architecture

If you are familiar with Apache 1.x, then very few things in Apache 2.0 will surprise you. For starters, Apache continues to be highly modularized, allowing you to include only those modules that you deem necessary in your server. But whereas Apache 1.3 had a core module that included the basic HTTP implementation, Apache 2.0 has delegated even more supported protocols to modules. This has a number of advantages, including the fact that we can now add (and subtract) protocols as necessary from Apache. In other words, Apache has now become a general-purpose internet server, rather than just an HTTP server. How many projects will take advantage of this functionality remains to be seen.

Apache was never meant to be the fastest server on the planet. Rather, it was designed to be extensible via a system of modules. Each module provided a different piece of functionality; administrators interested in squeezing the last

ounce of power from their systems don't have to include irrelevant modules. For example, if we know that our server will never run any CGI programs, then we can easily remove mod_cgi, gaining some CPU cycles and memory in the process.

Apache 2.0 continues in the long-standing Apache tradition of handling each HTTP transaction in a number of named phases. A module may examine or modify the transaction during any one of these phases by attaching its own handler to the appropriate hook. For example, mod_speling (which corrects capitalization and spelling mistakes in URLs—the name is purposely misspelled) attaches its handler to the “fixup” phase hook, executing immediately before the server generates a response.

In Apache 1.x, only one handler could fire for a given hook. In Apache 2.0, each handler not only registers itself for a given hook, but indicates when it would like to execute relative to other modules; mod_speling, for example, registers its handler as the final (APR_HOOK_LAST). If another module were to register with the fixup handler, it would execute before mod_speling. The fact that multiple handlers can fire for a given hook opens a world of possibilities that were previously too difficult to achieve.

On a similar note, Apache now makes it possible for one module to filter, or modify, the output of another module. This is currently possible with mod_backhand, but that module depends on a number of tricks and dark corners in the Apache API. Apache 2.0 is designed to allow modules to act as input or output filters. This means that if you want to add a standard set of headers or footers to your HTML pages, you can now do this across the board, including for dynamically generated pages created by CGI programs, server-side includes and mod_perl handlers.

The Apache configuration system now uses GNU autoconf rather than the Apache-specific system that was in use for versions 1.x. And, many of the C-language abstractions (such as hash tables and strings) that were included in previous versions of Apache have now been named the Apache Portable Runtime (APR). The APR is included with Apache and is configured and compiled into the server automatically when you build it.

Finally, Apache now comes with mod_ssl, which provides SSL and TLS encryption. Not only did Apache 1.x fail to come with such a module, but the two modules (Apache-SSL and mod_ssl) were incompatible and required patching the Apache source code before installation. The fact that mod_ssl will now be a standard part of every Apache installation is a huge relief for web site administrators and is most welcome.

MPMs

UNIX systems have long had the ability to run multiple processes simultaneously. I typically run Emacs, a GNOME terminal and Galeon on my Linux box; while a casual glance might only reveal these three processes, there are actually dozens more (sendmail, gnome-panel, Apache, syslogd and the like) that are executing without my direct knowledge. For a complete list of what is running on my computer, I can use the command **ps aux**.

The good news is that the process model is simple to understand, ensures stability on the system and is portable across many operating systems. Unfortunately, however, processes are relatively heavy and slow. Linux users are especially spoiled on this front because creating a new process on Linux is a surprisingly lightweight operation. But even on Linux, spawning a new process can sometimes be a bit extreme.

For this reason, an alternative model of threads has grown over the years. Using threads, a single process can be executing in multiple places at the same time. Threads offer many of the benefits of processes without the overhead. But there is a cost: programming with threads can be extremely tricky because it's always possible that a particular piece of code is executing in two different threads. You can always write (or rewrite) code to be threadsafe, but this is often a difficult task.

Because threads were both difficult and tricky to handle, and because Apache was originally designed to work only on UNIX machines, Apache 1.x worked exclusively at the process level—if you want to handle ten simultaneous HTTP requests, then you must have ten Apache processes running. Because it takes time to create a new process, Apache 1.x took an idea from NCSA HTTPd, preforking processes before they are actually needed. This means that Apache can be a bit slow to start up, but that handling the incoming connections does not take much time. Apache also allows administrators to indicate how many “spare servers” should always exist, adding and removing Apache processes as necessary.

Preforked Apache servers are solid, well understood and robust. But on many systems, using processes is inferior to threads. In particular, Windows uses threads far more than processes, which means that by sticking with processes, Apache was limited in its ability to penetrate the Windows market.

Apache 2.0 solves these problems with MPMs (multiprocessing modules). Each MPM is an Apache module that handles the details of processes and threads. On Windows, OS/2 and BeOS, this means that you can finally run Apache using a threading mechanism that is native to your operating system. On UNIX and

Linux systems, you can experiment with a number of different models, choosing one that is appropriate for your needs.

The prefork MPM, which runs in exactly the same way as Apache 1.x did, is the default choice when you install Apache. Two other choices for Linux users are: 1) worker: the number of threads rises and falls (according to the number of incoming requests), but the number of processes remains constant; and 2) perchild: each process contains a fixed number of threads, but the number of such processes rises and falls according to the number of incoming requests.

It's too early to tell, but I expect that more MPMs will emerge over time, and that there will be numerous modules that take advantage of threads to pool database connections, share application data and spawn asynchronous tasks in the background.

Configuring and Installing Apache

Retrieve the latest source code from httpd.apache.org/dist/httpd; the latest version as of this writing is 2.0.35. Unpack the source code in a temporary directory:

```
cd /tmp
tar zxvf httpd-2.0.35.tar.gz
```

You may now run the configure program with one or more arguments. These arguments fall into roughly four categories: 1) Into which directories should Apache be installed? 2) Which MPM do you want to use? 3) Under which user ID should CGI programs execute? 4) Which modules do you want to install? And of those, which should be installed dynamically (using shared libraries) rather than statically?

You can get a full list of configuration options by typing **./configure --help**. This is particularly true if you want to include a module that isn't included by default. The biggest change in configuration is that modules now have their own options to activate them. For the simplest possible configuration that uses the "worker" MPM, type:

```
./configure --with-mpm=worker
```

Following this, run **make**, followed by **make install**. (There is no **make test** for Apache as of the time of this writing.) By default, Apache 2.0 is installed into /usr/local/apache2. You can start it using the same program as Apache 1.x, apachectl, which is normally in /usr/local/apache2/bin/:

```
/usr/local/apache2/bin/apachectl start
```

The server will soon start up. HTML documents will normally be kept in /usr/local/apache2/htdocs, so you should already be able to put HTML documents there and view them.

Apache's runtime configuration remains dependent on a text file, normally called httpd.conf. If you are familiar with Apache 1.x, then you will be happy to know that almost all of the existing directives will continue to work. The main directives that you will have to learn are those that pertain to threading, assuming that you use the worker or perchild MPMs.

Should I Switch?

When the Apache Software Foundation announced Apache 2.0, the announcement explicitly said that the new version is the most stable and recommended version for production use. And for the most part, I believe them; www.apache.org receives many more requests per day than my server, and they have been running Apache 2.0 beta versions for over a year. Thus, it's safe to say that Apache 2.0 is stable enough for most sites to use.

The main reason to avoid switching to Apache 2.0 at this point is if you need mod_perl or PHP; they are currently still in testing but will probably be available by the time you read this.

As I mentioned above, however, it's hard to get threading right, and this is particularly true in Perl, which has experimented with a number of threading models in the last few years. If you have compiled Perl with ithreads, then you can use it to create a mod_perl for Apache 2.0 that uses the worker or perchild MPMs. But just how stable this configuration will be remains to be seen; it may well be that mod_perl users will choose to stick with the prefork MPM for now, until the dust settles a bit.

Conclusion

Apache 2.0 comes with more of everything that a web developer would want—more modules, more flexibility and greater speed. If you haven't yet tried Apache 2.0, I encourage you to download it and test it with your site's configuration to verify that it will be a good choice.

Resources

email: reuven@lerner.co.il

Reuven M. Lerner is a consultant specializing in web/database applications and open-source software. His book, Core Perl, was published in January 2002 by Prentice Hall.

Art Is but Engineered Reality

Marcel Gagné

Issue #99, July 2002

Generating your own landscapes, moonscapes and sheepscapes right on your desktop.

Cooking with Linux

Art Is but Engineered Reality

Generating your own landscapes, moonscapes and sheepscapes right on your desktop.

by Marcel Gagné

Dare I say, François, that this 1999 Côte-Rôtie is transcendent? Sipping a fine wine like this, I am transported to new planes of existence. Of course not really, François, it is only an expression. It is a compliment that I pay the vintner for such a fine wine. Am I not still here with you in this very restaurant, looking at the menu for this month's engineering focus issue?

Reality and perception are uniquely personal experiences, *mon ami*, experiences all Linux users may explore if they wish. One of the great joys of the Linux environment is the freedom it provides to talented programmers the world over to create alternative ways of working and playing. Those of us who merely use, prepare and serve their creations can then share in those visions as well.

François, where are you, *mon ami*? Your attention seems drawn elsewhere. Ah, *mes amis*! Welcome to *Chez Marcel*. I did not realize you were already here. François, *Vite*! Fetch more of this wonderful wine for our guests *immédiatement*. Please sit, everyone. Be comfortable. François will return momentarily with more of the 1999 Côte-Rôtie I was just sampling, er, I mean, subjecting to quality control, of course. It may seem strange, *mes amis*, but François and I were just discussing the nature of reality and perception.

I have never truly gotten over my first view of a great fractal landscape. The breathtakingly real mountainous panorama, with its snowcapped mountains and nestled lakes, made me understand the sheer beauty of mathematics. If something as complex as a landscape or a fern or a galaxy could be reduced to a series of mathematical symbols replayed over and over again, there is definitely something starkly beautiful at the heart of the universe.

Using a little program called xmountains, you can recreate those dreamlike landscapes for yourself. Go to author Stephen Booth's web site (www.epcc.ed.ac.uk/~spb/xmountains) and get yourself the latest version of his program. Engineer a little reality, manipulate it and dress up your desktop.

With that, it is time to extract and build the source. Notice that I have created a temporary directory into which the source can be extracted. The files are, unfortunately, not saved within their own directory. Building xmountains consists of these simple steps:

```
mkdir temp_dir
cd temp_dir
tar -xzvf xmountains_2.6.tar.gz
xmkmf
make
```

Then you can copy the resulting binary (called xmountains) to wherever you see fit (perhaps /usr/local/bin). There are many ways to tweak the program's output with command-line switches, but in its simplest form you can just type **xmountains &** and watch the show. A full-screen window will open up with a randomly generated landscape. Every ten seconds the landscape will scroll a little bit to the left with newly generated terrain. To see the available options, use the -h option. Have a look at Figure 1 for a sample of xmountains' output.

Figure 1. A Mountainous Virtual Reality

Depending on the window manager you are using, it is possible to modify xmountains to write directly to the root window, thus providing a constantly changing fractal landscape. By default, the code does not allow for this, and the effect doesn't work well on all window managers. Consequently, the author did not make this option a default for the program. I decided to go ahead and make the changes. Perhaps, *mes amis*, you will try it as well. The change is an easy one. Modify your Makefile and look for the line that reads **PROJECT_DEFINES =**, and change it to look like this:

```
PROJECT_DEFINES = -DVROOT
```

Now do a **make clean** followed by a **make** and rerun the program with the -b switch. The resulting pixmap will be written to your desktop's root window. The results under KDE weren't great, but everything worked beautifully under Window Maker. As they say, *mes amis*, the mileage, it varies, *non*?

Creating virtual landscapes is just one of the many applications of creating art with fractals. If you would rather just get in there and start generating a little eye candy, you may not need to install any software whatsoever. Most Linux distributions come with a package called xscreensaver that includes a collection of, well, screensavers. What you may not know is that each screensaver is a program unto itself with its own capabilities. For instance, drift is a program for drawing recursive fractal flames. It can be called from the xscreensaver program or directly, like this:

```
/usr/X11R6/lib/xscreensaver/drift -background "Wheat"
```

This will open a window with drift endlessly generating fractal flames on a wheat-colored background. The default is black but you can play with the colors. If you would like a larger window, (for instance, 800 × 600) try adding the -geometry 800x600 flag. Remember that talk about the root window when we discussed xmountains? As far as window managers go, the same rule applies here, but you certainly can launch the program to generate dynamic art on your desktop with the -root flag:

```
/usr/X11R6/lib/xscreensaver/drift -root
```

Do you long for the call of nature? A little joke, *mes amis*. Perhaps mountains are not your thing but, rather, the untamed beauty of the forest. Maybe a little window in the corner of your desktop is in order—try forest from the same package. The path to the executable will vary depending on your installation. On my test system, these all live under /usr/X11R6/lib/xscreensaver.

```
forest -geometry 300x200
```

When you are tired of this, try hopalong and experiment with fractal planes. You might also call up julia for some animated julia set fractals. What is that,

you ask? Well, according to the documentation, the program “uses ifs {w0 = sqrt(x-c), w1 = -sqrt(x-c)} with random iteration to plot the julia set, and sinusoidally varied parameters for the set, and plots parameters with a circle.” (Aren't you glad you asked?) To round out the xscreensaver offerings for today, consider one of my personal favorites, imsmap. Figure 2 offers a view of my rather busy desktop with many of these programs running simultaneously.

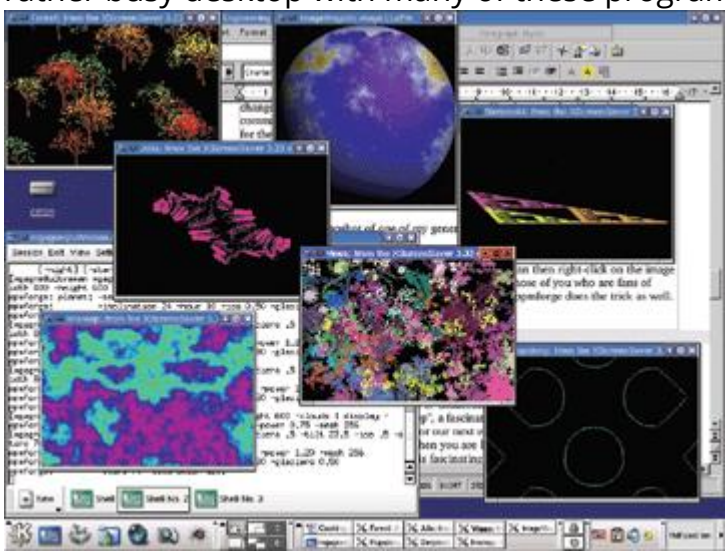


Figure 2. Fractals, as far as the eye candy can see.

Your Linux system is a treasure trove of programs when it comes to creating virtual landscapes of surrealistic art. Another package that was likely installed as part of your distribution is netpbm or netpbm-progs. As part of that package, you'll find a program called pgmcrater that will create a cratered terrain using fractals, so that you end up with a virtual lunar landscape (as in Figure 3). Try this:

```
pgmcrater -x 800 -y 600 > crater.out
convert crater.out crater.jpg
```

The second command is part of the ImageMagick toolset. The pgmcrater program generates images in a raw portable graphics format that gets dumped to a temporary file. I then convert the raw image into a JPEG, a format I can use to set a background image for my desktop. Incidentally, we could make those two commands into one with the following:

```
pgmcrater -x 800 -y 600 | convert - crater.jpg
<center>
```

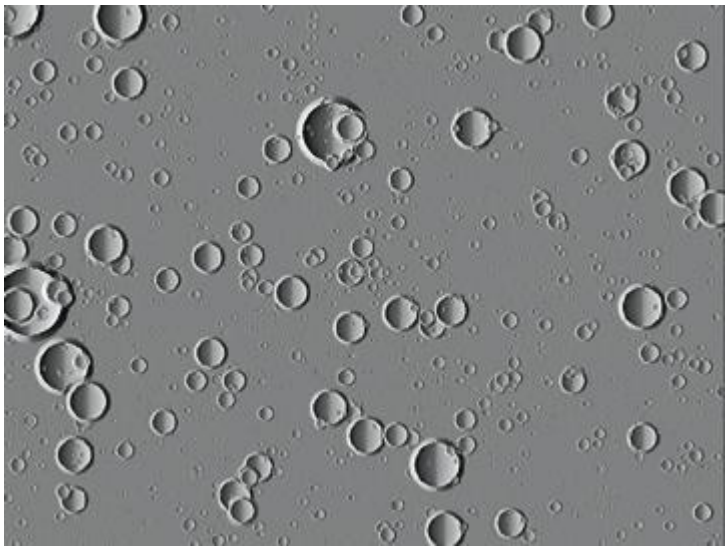


Figure 3. Is it the moon or pgmcrater

Before we move away from netpbm-progs, let's look at one more program, and one I am particularly fond of, ppmforge. This program generates “fractal forgeries” (a term coined by Richard F. Voss) of planets, starry skies and clouds. The images are incredibly realistic (as real as a computer-generated planet can be) and make for beautiful artwork. The program allows you to specify the amount of glacial formation (based on terrain altitude), the angle of tilt for your planet (in relation to the planet's star), the amount of polar ice and the time of day (which changes the position of the day/night meridian). Here's a sample:

```
ppmforge -hour 10.5 -glaciers .5 -tilt 23.5 -ice .5
-width 800 -height 600 -stars 70 | display -
<center>
```

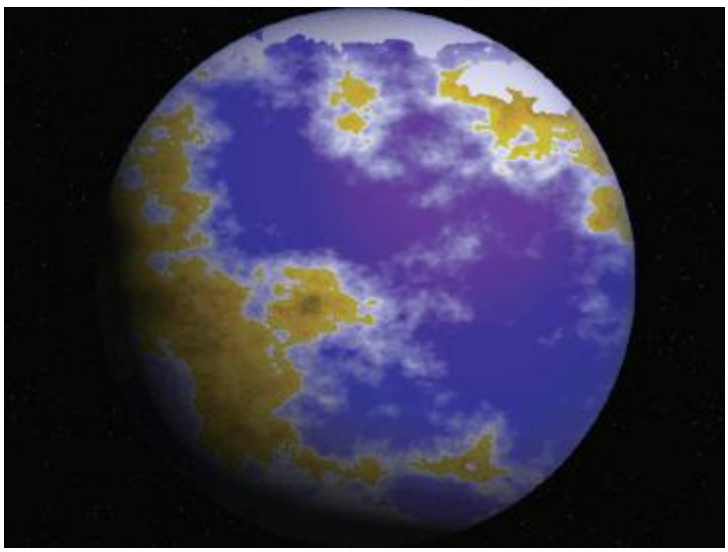


Figure 4. Strange, new worlds are yours to create!

Figure 4 is a snapshot of one of my generated worlds. The resulting image is being piped through to ImageMagick's display program rather than converting it to a fixed image each time. I can then right click on the image and choose to save it in whatever format I desire.

For those of you who are fans of backgrounds comprised of blue sky dotted with clouds, ppmforge does the trick as well. Try this version of the command:

```
ppmforge -clouds -width 800 -height 600 | display -
```

As you can see, inspiration is as close as your desktop. François, our guests are thirsty. Please, more wine.

One of your humble chef's favorite movies is *Bladerunner*, based on Philip K. Dick's novel *Do Androids Dream of Electric Sheep?*, a fascinating picture of reality if ever there was one. Dick's novel is the inspiration for our next encounter with alternative reality, Scott Drave's electric sheep. When you are lost for inspiration, pull your hands away from the keyboard, wait for this fascinating screensaver to kick in and prepare for a little electronic dreaming.

Here is the premise: when the screensaver starts, your screen will blank for a moment as the program connects to the server. Then, an animated “sheep” will appear. This sheep is actually an animated fractal frame created by the collaborative processing efforts of networked clients. New sheep (the resulting animation) are created every 15 minutes, 24 hours a day. It's hard to say that this is as useful as the Seti@HOME Project (another part of electric sheep's inspiration), but it is certainly very cool, *non*?

The nature of the high-resolution animations generated pretty much demands some kind of high-speed internet connection; a 56K modem may not be sufficient. If you are hungry to try it, may I recommend that you begin with a visit to electricsheep.org for your copy. Then, extract and build it like this:

```
tar -xzvf electricsheep-2.1.tar.gz.html
cd electricsheep-2.1
./configure
make
su -c "make install"
```

You can start the program and test it without going into screensaver mode by simply typing **electricsheep**. In this mode, it will occupy the root window background, so you may have to minimize your active windows. You can also pass the --help flag for a list of options. To use it as a screensaver with the xscreensaver package, modify (or create) your .xscreensaver file with electric sheep listed as one of the programs. In the following example, I have also set my screensaver timeout to two minutes:

```
timeout:           0:02:00
lock:                True
lockTimeout:    0:02:00
programs: \
   electricsheep --nick "ChefMarcel"   \n
```

Then, start up xscreensaver with the command **xscreensaver -nosplah &** and wait. Figure 5 shows off one of the many sheep that appeared on your humble chef's screen.

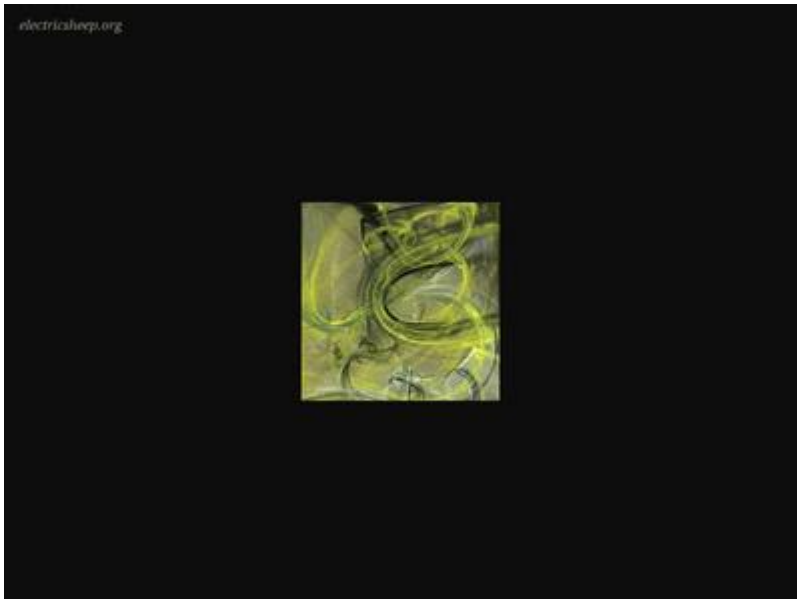


Figure 5. Does your Linux system dream of electric sheep?

Speaking of dreams, *mes amis*, the hour grows late and we must soon close the restaurant. In order to help you sleep more comfortably tonight, François will refill your glasses a final time. François, *s'il vous plaît*. As usual, your visit to *Chez Marcel* has been a real pleasure. Dare I say *dreamy*? *Non*.

Finish your wine. Enjoy. Until next month. *A votre santé*! *Bon appétit*!

Resources

Marcel Gagné (mggagne@salmar.com) is president of Salmar Consulting Inc., a systems integration and network consulting firm and the author of Linux System Administration: A User's Guide, published by Addison-Wesley.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Staying Current without Going Insane

Mick Bauer

Issue #99, July 2002

How tools like SuSE's YaST2, Red Hat's up2date and Debian's apt-get can help you maintain some measure of patch sanity.

Security is a moving target and chasing it can be positively dizzying. One of the most unrelenting and tedious security tasks on powerful systems like Linux is keeping its myriad applications, commands and libraries current. As anyone who subscribes to the BugTraq mailing list can attest, new software vulnerabilities are constantly being discovered, exploited and patched against. But given the hundreds of software packages installed by the typical Linux distribution, how can you ever hope to keep up?

The bad news is, you can't. Even if you had a way of instantly patching against every new vulnerability published on every vulnerability and incident report mailing list, sooner or later you would become the subject of an incident report yourself; some vulnerabilities don't become common knowledge until they've been exploited.

The good news is that since you can't possibly keep up, any progress you do make is a win. If you get cracked via a three-week-old software vulnerability, at least you're more elite than someone who gets cracked via a three-*year*-old 'sploit. (On the other hand, either way you're cracked.)

In all seriousness, though, from a purely statistical standpoint, fewer unpatched bugs means fewer exploitable vulnerabilities. Despite how thankless and endless the task seems, it *is* worth trying to keep your Linux software current. And luckily, recent versions of popular Linux distributions include new tools for automating much of this task, including SuSE's YaST2, Red Hat's up2date and Debian's apt-get. (Some of these tools are even secure!)

Packages vs. Source

I'll start with a key piece of advice that I've arrived at after years of skepticism: wherever possible, stick to your distribution's supported packages. To many people, this probably sounds obvious; why build from source if you don't have to? But to some of us old-school types, building from source is as much a habit as a skill. Maybe this is because way back in the early 1990s (you know, back before we had Slashdot or powered flight), there was a lot less software for Linux than there is now, which meant that many of the things we ran we had to build from sources developed on other platforms. Additionally, the whole concept of packaging Linux into distributions was much less mature: there were fewer distributions, and they didn't change nearly as frequently.

(Have you ever had to compile ps from source because your brand-new kernel wasn't backward-compatible with the older version of ps from your distribution? Some of us have. We also had to walk ten miles to school each day through snake-filled swamps, etc.)

But things are different now. Don't get me wrong: I don't mean that we should all be slaves to binary packages. Sometimes you need features that are available in the very latest version of an application but not in your distribution's version of it; sometimes you want a leaner-and-meaner build than the one-size-fits-all juggernaut that your distribution provides. However, there are some important advantages to sticking with binary packages most of the time.

The first is convenience: downloading a large group of applications from a single site is faster and easier than downloading each from its developer's site (which is why we have distributions in the first place), and installing a binary package is much faster and less prone to mishaps than compiling from scratch. Convenience is not something we *nix bigots admit to valuing, but there it is.

The second advantage of packages is stability: the major distribution packagers put a good deal of testing and research into deciding whether to include a given application in their distribution, and if so, which version is the most stable and which compile-time options best suit their distribution. There have been notable exceptions to this, but most distributions nowadays do quite well with quality assurance.

Stability is unquestionably a big factor in security: where there are bugs there are vulnerabilities. Even bugs that don't have obvious security ramifications often can be exploited in, for example, denial-of-service attacks (the object of which is to crash a system, which some bugs do effectively).

This leads us to one of the more frustrating paradoxes in application security: although some of the most widespread vulnerabilities on the Internet stem from poorly maintained applications (i.e., obsolete and/or known-vulnerable versions), newer is not always better. The security community has rightfully lambasted Microsoft over the years for being slow to acknowledge and provide patches against security vulnerabilities in their products. But we complain equally bitterly when Microsoft *does* provide a speedy patch that affects stability because this potentially mitigates any benefit derived from fixing the original bug.

Ignoring for a moment stability's desirability in its own right, ask yourself this: supposing an application has an obscure buffer-overflow condition that is theoretically exploitable for root privileges (but only by a skilled assembly programmer with a working knowledge of the RC4 stream cipher), and you patch it with code that introduces a denial-of-service opportunity that can be exploited even by attention-span-deficit script kiddies—how much security have you really gained? The answer to this will vary depending on the precise circumstances and on who you ask, but the important point is that software upgrades often have ramifications of their own.

I'm beating this point to death, but it's an important point because it debunks the notion of instant software updates being some sort of panacea. (Plus it's been on my mind for a long time; it's taken me years to fully understand why, for example, the very latest version of OpenBSD still ships with a hack of BIND v.4.9.8, which in computer/dog years is really ancient.)

It's also important because, getting back to my original topic, there's something to be said for letting your Linux distributor make the difficult patching decisions, and thus for waiting to patch a vulnerable application until your distribution releases an official (and hopefully tested) patch, which of course is part of the deal when you rely on binary packages rather than cold hard source code. It's difficult enough to keep up with Linux distributors' application updates; I can only imagine how futile it would be to patch *and recompile* all my critical applications myself and waiting anxiously to see whether the patching broke anything else.

So to summarize these pearls of wisdom (at least I hope they're pearls, instead of some other small spherical secretion): keeping current is a virtue, using binary packages is a virtuous form of laziness (props to Larry Wall) and relying on secondhand security updates from your Linux distributor rather than “going cowboy” isn't laziness at all, it's prudence.

Which Systems Should You Update, and How?

Before I talk about automatic update tools, let's talk about different kinds of systems for a moment. A bastion host connected to the Internet has a much more compelling need for the immediate application of security updates than a desktop system behind a firewall. Use common sense in deciding on which systems you're going to expend the necessary time and effort to keep strictly current, and for which systems you're willing to sustain a little risk by not being as painstaking.

Another thing to consider with regard to systems' roles is which hosts you can comfortably run an X-based update tool on such as YaST2. Bastion hosts (publicly accessible ones) shouldn't run X, generally speaking. If you really like YaST2's ease of use and overall convenience, run it in an internal system, configure it to save updates and push those updates out to your bastion hosts via scp.

Red Hat up2date

Starting with Red Hat 6.2, you can use up2date to identify, download and install updated packages automatically. **up2date** runs in either text or X mode, so the “where may I run it?” conundrum described above doesn't really apply.

Before you can run up2date, you need to configure it. Luckily, there are two convenient and simple tools to do this. First, you need to use rhn_register to create an account on the Red Hat Network (RHN). If invoked without arguments, rhn_register runs in X mode, but you can run it in text mode by passing it the --nox flag:

```
bash-# rhn_register --nox
```

You can register one system with RHN for free, but subsequent systems registered to the same account require a fee. RHN registration is necessary for every host on which you wish to run up2date. There's nothing to stop you, however, from running up2date on one machine, saving the updated packages and pushing them to your other Red Hat systems, via scp or some other secure means, and installing them on those systems manually.

Another thing you need to know about RHN is that by default, a system registered with RHN will send RHN information about its network configuration and hardware, plus a complete list of all installed packages and their versions. This allows you to schedule automated updates to be sent from RHN to you and/or for customized e-mails to be sent to you whenever an update is available for packages installed on your system.

Those two RHN features may be very appealing to you, and if either makes the difference between your performing regular security updates, I for one won't criticize you. However, I personally am uncomfortable providing detailed lists of my system's setup to strangers without needing to. It isn't that I have any specific reason not to trust the fine professionals at Red Hat Network; it's just that I'd rather not *have* to trust them.

After all, it's really no big deal to me to subscribe to the Redhat-Watch-list, which Red Hat uses to announce new updates (listman.redhat.com), and to decide for myself whether it's necessary to run up2date. If it is necessary, up2date will automatically determine which updates are applicable to my system even if I haven't stored any system information at Red Hat. Therefore, my own practice (and recommendation) is to deselect the rhn_register options “Include information about hardware and network” and “Include RPM packages installed on this system in my System Profile”.

After you've registered your system with the Red Hat Network, you need to run up2date-config (Figure 1). Like rhn_register, this command supports the --nox flag if you wish to run it in text mode. **up2date-config** is for the most part self-explanatory, but several settings are worth mentioning.

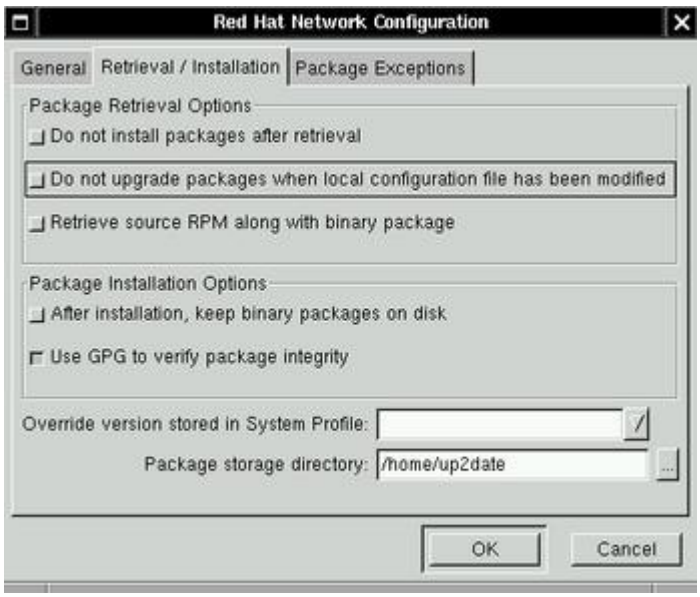


Figure 1. Red Hat's up2date-config Tool

First, if you're going to download, store and distribute updates from a central system as described above, you'll want to select the option “After installation, keep binary packages on disk”, and specify (or at least note) the default value of “Package storage directory:”. If you run rhn_register with the --nox flag, these options are instead labeled keepAfterInstall and storageDir, respectively.

Second, be sure not to deselect the option “Use GPG to verify package integrity” (it's selected by default) unless your system is very slow. One of the niftier

features of the RPM package format is its support of internal GPG signatures that can be used to verify the integrity of the RPM. **up2date** can do this automatically for you, provided GnuPG is installed on your system. If it is, you can also verify an RPM package's GPG signature manually like this:

```
rpm --checksig /path/packagefilename.rpm
```

where, naturally, you should replace */path/packagefilename.rpm* with the full path of the RPM file you wish to check.

Sooner or later somebody will crack a Red Hat mirror site and replace some essential package with a trojaned version; when that happens, those of us who routinely check for valid RPM signatures will be far less likely to be stung.

Once you've registered with RHN and run up2date-config, you can run up2date itself (Figure 2). There's not a lot to say about this; the whole point of up2date is simplicity and convenience, so you don't need me to tell you how to click its clearly labeled buttons.

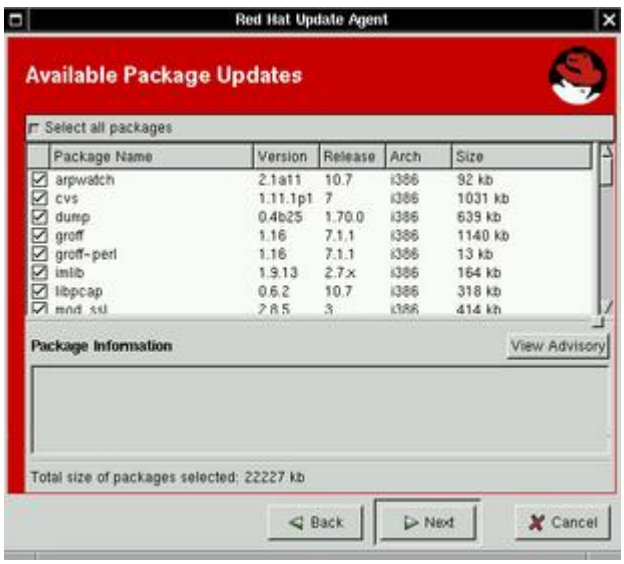


Figure 2. Part of an up2date Session

I will repeat, however, my earlier advice: sign up for the Redhat-Watch-list and run up2date whenever an update is announced that is applicable to your system. By using such a slick and user-friendly tool, you have little excuse not to.

SuSE's On-Line Update

If you use SuSE Linux, you've got an even easier-to-use tool at your disposal for automated updates: YaST2, with its on-line update module (Figure 3).

Figure 3. Running On-Line Update from YaST2

Unlike Red Hat's up2date, you don't need to register with SuSE to run this, nor do you need to use or edit a separate file to configure it. The on-line update is self-contained, and in the first couple of screens of an on-line update session you can change its configuration if needed (e.g., you can select a download site that's geographically closer to you than SuSE's US-based ftp.suse.com server).

In order to know when to run the on-line update, you should subscribe to SuSE's suse-security-announce mailing list. Like the Redhat-Watch-list, this is a low-volume e-mail list, so don't worry about SuSE spamming you with frivolous notes. To subscribe see www.suse.com/en/support/mailinglists/index.html.

Honestly, there isn't much else I need to tell you about YaST2/on-line update, except for one minor problem I've had with it (an end-user error, actually, but an easy one to make). If you invoke the command yast2 manually from an xterm or a “Run command” dialog and your X session isn't being run as root, the on-line update will fail, returning a misleading error about being unable to locate the update list on the specified FTP server.

This isn't the case; actually, YaST2 needs to run as root in order to write this file after obtaining it from the FTP site. That doesn't mean you must run YaST2 only from a root X session; it means that if you don't, you should use the menu item automatically created by SuSE for YaST2, in which case you'll be prompted for the root password and the on-line update will work properly.

Manually Updating RPM Packages

In my enthusiasm for up2date and YaST2, I haven't yet mentioned one other simple method for updating RPM files: the rpm command itself, which works

equally well on Red Hat (and its derivatives) and SuSE. The easiest way to illustrate this method is with an example.

Suppose you receive notification of a vulnerability and available update for the fictitious SuSE or Red Hat package blorpflap, and you use the URL provided in the notice to download the updated RPM to the local path /usr/pkg/updates/blorpflap-3.2-3.rpm. First, you should verify its validity:

```
rpm --checksig /usr/pkg/updates/blorpflap-3.2-3.rpm
```

Naturally, for this to work, your distribution's signing key will need to be on your GnuPG public keyring. See my two-part series on GnuPG in the September and October 2001 issues of *Linux Journal* for a tutorial on using GnuPG.

If the GPG signature checks out okay (or if you assume that it does, which you're free to do at your own risk—the above step is optional), install the update:

```
rpm -Uvh /usr/pkg/updates/blorpflap-3.2-3.rpm
```

-U is short, of course, for update (actually update or install), and it works for both updates to previously installed packages and for new packages; -v makes the action verbose, which I like; and -h tells rpm to print out a little progress bar.

How Often Should I Upgrade the Entire Distribution?

Debian's apt-get

The last tool we cover this month is Debian's apt-get. As seems to be characteristic of Debian, apt-get is less flashy but in some ways even easier to use than other distributions' fancy GUI-driven equivalents. In a nutshell, there are only two steps to updating all the deb packages on your Debian system: 1) update your package list and 2) download and install the new packages. Both steps are performed by apt-get, invoked twice:

```
bash-# apt-get update
bash-# apt-get -u upgrade
```

The second command tells apt-get to use wget to download updated packages and then to install them.

To receive e-mail notification of Debian security vulnerabilities and updates, subscribe to debian-security-announce by filling out the on-line form at www.debian.org/MailingLists/subscribe. Run apt-get whenever you receive notice of a vulnerability applicable to your system.

By the way, as much as I love apt-get, there is one important feature missing: GPG signature validation. This is because unfortunately, the deb package format doesn't support signatures and because Debian packages aren't presently distributed with external signatures. Reportedly, a future version of the deb format will support GPG signatures.

That's it for this month. Good luck keeping current and staying sane!

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

Archive Index Issue Table of Contents

Advanced search

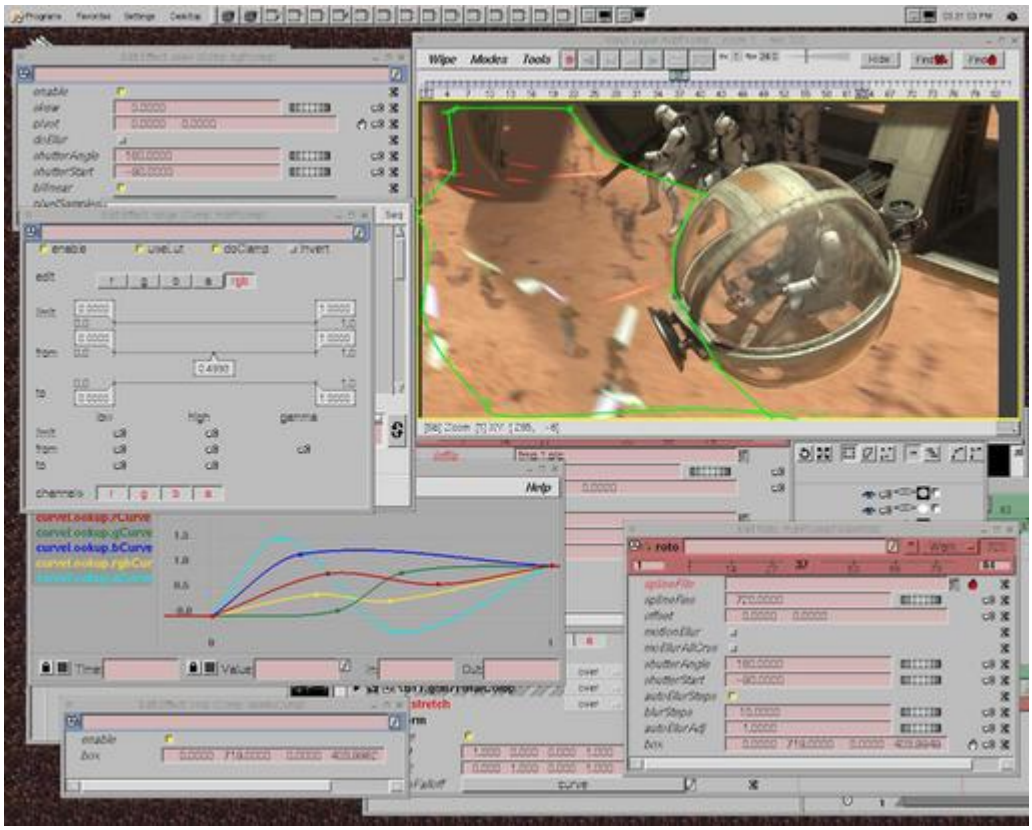
Industrial Light and Magic

Robin Rowe

Issue #99, July 2002

Discussing the move to Linux on ILM's renderfarm, with speed and stability comes responsibility.

Star Wars, Episode II: Attack of the Clones, released in May 2002, is Industrial Light & Magic's (ILM) first movie produced after converting its workstations and renderfarm to Linux last year. Located north of San Francisco in San Rafael, California, ILM was founded in 1975 to produce the visual effects for *Star Wars*. Although owned by George Lucas, ILM produces visual effects for more than Lucasfilm productions, such as the *Star Wars* and *Indiana Jones* films. Many other studios seeking that bit of something extra on the cutting edge of special effects use ILM. ILM has received 14 Academy Awards, including ones for its work on *Forrest Gump*, *Jurassic Park*, *Terminator 2*, *Who Framed Roger Rabbit?* and *E.T.*



A scene in *Star Wars* being edited in ILM's proprietary compositing package CompTime. ILM created its own compositor rather than using a commercial package such as Shake or RAYZ.

"Linux is increasing the quality of our work, not the quantity", says Andy Hendrickson, director of research and development. Large amounts of processing power enable more user control. He explains,

> We often go into a show knowing what we want but are forced to scale back realism with shortcuts because of a lack of processing power. Using Linux we can add more realism. We direct effects. It isn't enough to have a cloud that is an NOAA-accurate model. Artistic staff directs the effects with, "Make that cloud more fluffy". Or, if we simulate an entire ocean, as in *Perfect Storm*, "Make that wave larger".

ILM made a bold move to undertake their Linux conversion in the midst of a major movie production, switching while work was underway on *Episode II*. "We thought converting to Linux would be a lot harder than it was", says Hendrickson. "Linux is so like what we had before. We pushed forward deployment in November 2001 and will finish conversion after *Episode II*." During the changeover, ILM is supporting existing SGI IRIX machines and Linux PCs to avoid overwhelming users with too much change.

Sequence supervisor Robert Weaver is a technical director on *Episode II*. Weaver's desk has a Linux PC on the left side and an SGI O2 on the right. Because the Linux desktop is configured to look like the SGI O2, it isn't

immediately apparent which screen is which, until Weaver demonstrates the difference in speed. He says,

> The old system is so slow that the clones firing lasers appear to be throwing javelins. We've seen about a five times speed improvement with Linux, which is appreciated! I'd say Linux is one of the most successful efforts we've had. I can't say enough good things about it. It is intuitive, incredibly stable, and we can get stuff fixed at a moment's notice. Our renderer hasn't been ported yet but will be by the summer. That's all that's keeping me on the O2.

All ILM 3-D particle simulations are done in Alias|Wavefront Maya. “We have, I'd say, 90% of our Maya users on Linux”, says Weaver. “It seems incredibly stable on Linux. I haven't had Maya crash on me in months. I'm evaluating that the correct cycles have been put in. I do that in wireframe mode.” To extend the functionality of Maya, Weaver writes plugins. “Maya makes writing plugins fairly easy. I add stuff to the shelf.” The shelf is a set of plugin tabs visible across the top in Maya. The ocean in *Perfect Storm* is an example of the effects ILM achieves with Maya plugins.

“Our compositing software, CompTime, has been ported to Linux”, notes Weaver. ILM created its own compositor with a plugin architecture for doing motion picture editing rather than choosing a commercial package. Weaver writes compositor plugins, too. “The compositor plugins are in Python”, he notes. “We're a big Python shop...and MEL.” MEL is the Maya scripting language.

Maya is considered by ILM a tool best for TDs (technical directors); animators at ILM use SOFTIMAGE. The conversion to Linux triggered a company-wide upgrade from version 3.8 of SOFTIMAGE (on IRIX) to the 4.0 version that recently became available for Linux.

In the years since the first *Star Wars* trilogy, animation software has become capable of greater facial expression. ILM created their own caricature facial animation application that reads and writes SOFTIMAGE scenes directly, not as a plugin. Senior Digital Model Supervisor Geoff Campbell used this software to set up facial expressions for animation in *Episode II*.

ILM Model Supervisor Geoff Campbell and Animation Director Rob Coleman working on the character animation of Yoda for a scene in *Star Wars*.

“There are 11 muscles in the face that are key to giving a performance”, says Campbell. “I can stretch a face in SOFTIMAGE as much as I want. When I like what I've done, I'll save it as a new shape. At my desk I have a little camera and a mirror I use to view my real facial expressions. You invest a little bit of yourself in each character.” Campbell says an important detail in a character's performance is “eye darts”, the little telling looks that performers give each other when interacting. In *Episode II* Yoda had eye darts even with his eyes closed. “Linda Bell developed an animation of the eyes while sleeping, that is, the eyes moving in REM sleep under closed eyelids.”

“I wanted Yoda to look better than the puppet, to have the lip movements better match the words”, says Campbell. She explains that

> The interesting thing with Yoda is George didn't want us to go light years ahead of the puppet's limitations. When Yoda is speaking in *Episode II*, and he has a lot of lines, he uses phonemes in a very simplistic way. A lot of the shapes we are trying to mimic are the puppet shapes that Frank Oz created moving his hand in rubber.

The later CG Yoda matches the character in other movies in the series, but he has more exact lip phonemes, with the lips curling to make an “M” or “B” sound, than the puppet could create.

For Yoda in *Star Wars*, ILM used their proprietary facial caricature animation software designed to be file-compatible with SOFTIMAGE. ILM's approach to in-house development includes both proprietary applications such as this and CompTime, and custom plugins for use with commercial pacakges like Maya and SOFTIMAGE.

The hair on Yoda is another character feature manipulated with ILM's facial animation software. Because moving individual hairs would be too cumbersome, there are single control hairs that influence the hairs around them. To style Yoda's hair interactively, speed is important. When running ILM's facial animation software on the SGI O2, it took seconds to redraw the screen after each change, and the delay made work difficult. “With Linux we manipulate high-res models in real time in a way we couldn't with our SGI system”, says Campbell.

ILM still builds some physical models but mostly for backgrounds or for organic-looking things that can be created easier than with CG. Although ILM doesn't construct many spaceship models anymore, their computerized motion control cameras are still shooting background plates nonstop.

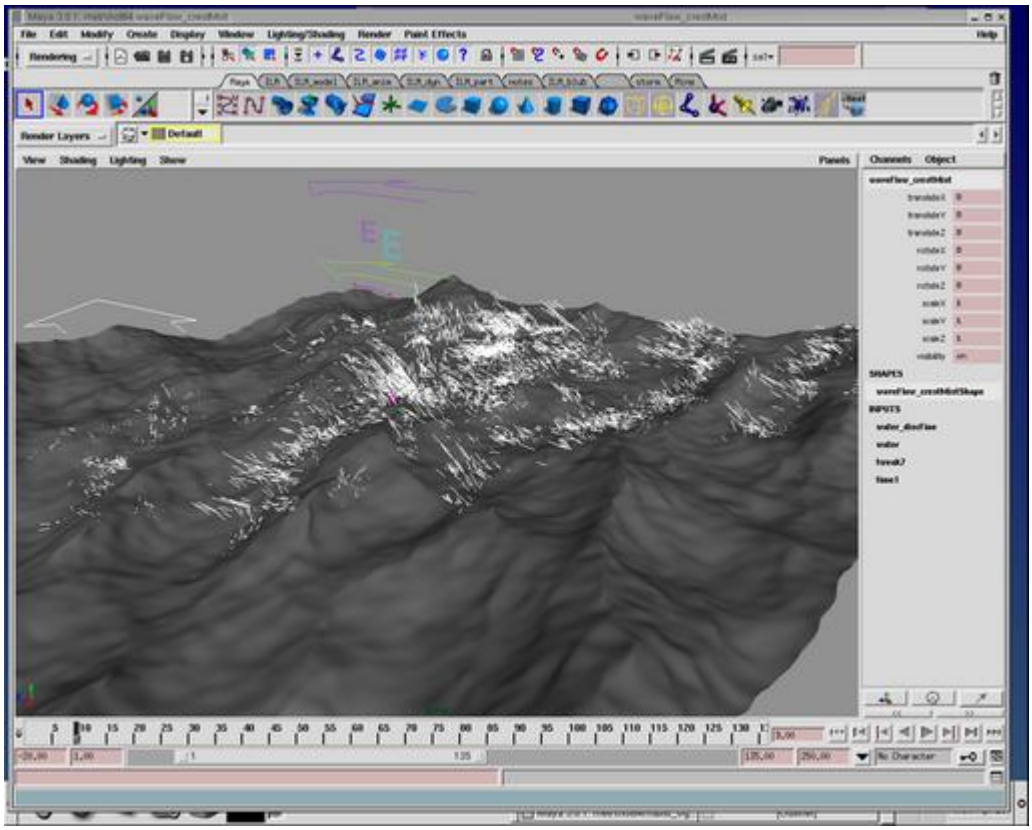
R&D Principal Engineer Phil Peterson reports that ILM is about 80% finished with its Linux software conversion. He says, “A team of three people ported over a million lines of code to Linux.”

“The biggest issue we had in porting was the compiler and other tools”, says Peterson. “Newer C++ code is fairly dependent on STL.” The gcc 2.96 compiler included with Red Hat didn't support the C++ Standard Template Library (STL),

so ILM uses gcc 3.01 instead. Their multiplatform build environment is customized based on Python cooperating with GNU make.

ILM had to accommodate some CPU differences, such as floating-point implementation and number precision. “In some cases we hand-optimized in-line assembly to get the most out of Linux”, says Peterson. One issue is how to track memory access per thread in Linux, which handles thread IDs differently from IRIX. Another annoyance is that a floating-point exception isn't allowed to throw a C++ exception (because FP exceptions are asynchronous).

In integrating legacy Motif applications, ILM had to overcome some issues with widget differences. “We were using Motif mostly”, says Peterson, “but use FLTK in our latest applications”. ILM made their SGI-based apps look similar on Linux, including the fonts and colors. ILM software projects may incorporate 80 or 90 libraries. For sound ILM uses OSS, but Peterson says they may switch to ALSA. SGI provides the dMedia libraries, but on Linux ILM had to create some of their own media libraries to fill in missing functionality. To play back movies, which at 2k by 1k are more than 27 times larger than typical 320 × 240 PC video, ILM created their own QuickTime-compatible library used in their flipbook player.



For the ocean in *Perfect Storm*, ILM created a Maya plugin with the capability of directing the intensity of particular waves. A key feature of ILM-developed software is user control.

“With Linux the increase in speed is what everyone is noticing”, Hendrickson says. He says the speed increase is

> ...not just 20% or 30%, but five times faster. We have an incredible appetite here for computation. It is the limiting factor. If an artist can get more iterations, the result will be that much better. We have lots of Jedi knights, lots of robes to model. Without the speeds coming out of hardware, we would not be able to do it. Before we had to hand-animate approximations. *Episode II* uses digital stunt double work, blending seamlessly between the live and the CGI performer. We are using CGI to replace rubber prosthetics on actors. We're doing more simulation, more rigid body dynamics. As droids get sliced and diced with light sabers, pieces fly off in realistic ways.

George Lucas, who used 400 shots in the original *Star Wars*, used 2,000 in *Episode II*. Creating that required three visual effects supervisors, as if doing three shows. “Expect a jump in what we're able to do after *Episode II*”, says Hendrickson.

Thanks, in part, to Linux.

The Computers of ILM

Resources

Robin Rowe (Robin.Rowe@MovieEditor.com) is a partner in the motion picture technology company MovieEditor.com. He has led video R&D at a Fortune 500 IT company, taught C++ at two universities and was an NBC-TV technical director. He leads two users' groups: LinuxMovies.org and OpenSourceProgrammers.org.

Archive Index Issue Table of Contents

Advanced search

On Engineers

David A. Bandel

Issue #99, July 2002

Here's one type of computer engineering we could do without.

Engineering—that word always implies engineers. In our case, the plural form of the word is appropriate. We have mechanical engineers that design the cases that house our systems (it's their job to ensure it takes at least half an hour to figure out how to open up any new case). We have electrical engineers that design faster, cheaper and better circuits, and we have systems engineers that design our systems, at least the software components thereof. I, however, want to suggest a new category: bloatware engineer. It would be the bloatware engineer's job to ensure that however much system you bought last week—say a P-IV 1.8GHz system with 512MB RAM, water-cooled, super-charged, over-clocked, speed demon—it would run like a 486-33 with only 16MB RAM.

My first nominations for this new category include the GNOME and KDE Project leaders. Personally, I use Blackbox (starts in about two seconds) and Sylpheed (starts almost instantaneously). The worst offender I run is Netscape, but it's the only fully CSS-2-compliant browser out there, and it's worse with the CrossOver plugin. I might also nominate just about any office suite, but bloat there is expected (although Applix always started and ran like a rabbit on methedrine). One day I'm going to go back and load a bunch of old software on a new system, just to see 1) if it even runs and 2) just how fast.

But don't get me wrong. I personally can't wait to buy a Cappuccino GG-1, the very small footprint computer available from Think Geek (thinkgeek.com), build a battery, get one of those new “keyboards” that's projected in front of you and senses where your fingers are and, finally, get those lightweight goggles that are like having a 20“ screen. All that's missing is a wireless card (need to swap out the Ethernet connection in the Cappuccino for a wireless connection). Talk about mobility in a three-pound package.

IP-Atlas www.xpenguin.com/ip-atlas.php

This is a small web script that takes an IP (your eth0 address by default) and plots it on a map of the world. The IP is sent to NetGeo, which returns a latitude/longitude reading of your supposed location. Overall, it's accurate but subject to assumptions, one being that the location of the company the IP is assigned to is the IP's ultimate geographic location. For example, the geographic location of one of my servers here in Panama shows up as in Texas. While my provider's parent is in Texas, and it's been pretty hot and dry down here up until now, this definitely isn't Texas. But it's fun anyway. Requires: web server with PHP and GD, internet access (to access NetGeo site).

iftop www.ex-parrot.com/~pdw/iftop

This is a top-like utility that watches your network connections on whatever interface you tell it to listen (by default, eth0). It runs on a VT (VC) or in an xterm window and shows current connections. For a way to quickly monitor what's being forwarded through your system in real time, this utility wins hands down. Requires: libpthread, libm, glibc.

OpenTRS otrs.org

The Open Ticket Request System (OTRS) is a web-based ticketing system that will take e-mail addressed to the OTRS user and allow you to queue it for resolution. The system has a link for a phone queue, but the ability wasn't yet available. This would, however, allow you to track calls and e-mails from customers or from systems sending you e-mails. The author has provided capabilities to respond via e-mail, to keep internal or public notes, etc. Requires: Perl; Perl modules: Unix::Syslog, CGI, MIME::Words, Mail::Internet, MIME::Parser, DBI, DBD::mysql, Digest::MD5; web server; SMTP (sendmail, etc.); procmail.

OpenGrade www.lightandmatter.com/ogr/ogr.html

This application for teachers sports both a text interface that is quick and easy to use, as well as a GUI interface. The GUI uses Tk, so you'll need that installed, but it's even easier to use than the text interface. While I didn't test it, if you're running a web server on the same system OpenGrade is running on, you can allow students password-protected access to their grades. Requires: Perl; Tk; Perl modules: Term::ReadKey, Date::Calc, Tk::FileDialog, Net::FTP.

dailystrips dailystrips.sourceforge.net

Okay, everyone needs a laugh or two, and a lot of folks I know open their local newspaper to the comic section first. With dailystrips you can have your

favorite web comics on one page waiting for you when you arrive at your desk. Requires: Perl, graphical web browser.

DACT www.rkeene.org/devel/dact

DACT is a dynamic adaptive compression tool that tries to find the optimum compression for a file based on block sizes. It also allows you to perform encryption on the file. Compression algorithms depend on the compression libraries found on your system, so not all compression algorithms will be available. Requires: libdl, libbz2, libz, libm, glibc.

NetSaint www.netsaint.org

Well, this month's pick from this column three years ago was just too easy. I have only one application I'm still using: NetSaint. NetSaint has come a long way since I originally reported on it. Installation is much easier, but unfortunately, while configuration is not difficult, it's not easy either. Requires: libm, glibc, web server, libgd (optional).

Until next month.

David A. Bandel (david@pananix.com) is a Linux/UNIX consultant currently living in the Republic of Panama. He is coauthor of Que Special Edition: Using Caldera OpenLinux.

Archive Index Issue Table of Contents

Advanced search

In Search of the Embedded Linux "Killer App"

Rick Lehrbaum

Issue #99, July 2002

Rick surveys internet-edge devices including next-generation set-top boxes, server devices and wireless access points.

Ask a sampling of embedded Linux vendors what they think the most likely embedded Linux "killer apps" are, and you'll probably end up with a list similar to this:

- Telecommunications infrastructure (high-availability, carrier-grade systems).
- Handheld mobile devices (including PDAs and smart phones).
- Wireless access points (for public 802.11 or Bluetooth WLAN access).
- Digital set-top boxes (digital TV, video and audio systems).
- Server devices (offering easy-to-use LAN-server/firewall/gateway/router/VPN functions).

Since the last three in this list—wireless access points, set-top boxes and server devices—are all small appliance-like "black boxes" that sit on the edge of the Internet and provide access to internet-based services and resources, let's combine those last three into one and call it "internet-edge devices". That simplifies the list into three categories of embedded Linux killer apps: 1) telecommunications infrastructure, 2) handheld mobile devices and 3) internet-edge devices.

Telecommunications Infrastructure

I've already covered the first of these in this column a few months back (see "Embedded Linux Targets Telecom Infrastructure" in the May 2002 issue of *Linux Journal*). The growing support for Linux in this space from telecom equipment heavyweights HP, IBM, Motorola and Sun Microsystems leaves little

doubt that Linux is going to have a bright future in the telecom equipment market—when the telecom market finally takes off, that is.

Handheld Mobile Devices

As for the use of embedded Linux in PDAs and other handhelds, success may well hinge on Sharp's Zaurus, the first such device to be strongly promoted by a world leader in consumer electronics. With the Zaurus just starting to make it into retail channels, it's too early to predict whether the Zaurus and its Linux-based software stack are going to be long-term winners. Rumors abound of ongoing projects to develop Linux-based PDAs and smart phones, particularly in Asia, so we can expect to see five to ten such devices hit the market during the latter half of 2002.

So far, embedded Linux has not fared well in the handheld mobile device market, with many projects being canceled and some resulting in stillborn products. Microsoft's Pocket PC, on the other hand, has grown steadily in market share at PalmOS' expense. Is it too late for embedded Linux in the handhelds market? The next six to nine months will answer that question.

Internet-Edge Devices

This brings us to the third killer app category: internet-edge devices. Think of these as highly compact, web-enabled server appliances having a variety of application-specific task assignments. Given the unquestioned success of Linux in the server space, coupled with the proven capability of Linux to satisfy the tight resource constraints of embedded devices, it should come as no surprise that Linux is likely to be a big winner in this important emerging class of products.

Let's take a look at some products in the internet-edge devices category that use embedded Linux as their internal OS platform.

Next-Generation Set-Top Boxes

This class of devices spans a wide range of capabilities, including such functions as personal video recorder functions, video-on-demand, e-mail and web access, and streaming video and audio media storage and playback. Here are a few embedded Linux-based examples:

- TiVo Personal TV: need I say more?
- Nokia Mediaterminal: this device adds a broad range of internet-based services to a normal TV set. Among the services available are digital audio/video, digital TV, video-on-demand, cached TV programs, web access, e-mail and chatting, games/gaming and many web-based software

applications. The Mediaterminal's internal software is based on Linux, the Mozilla open-source browser, the X Window System, plus a unique user-interface technology called Nokia Navi bars. The embedded computer is a 366MHz Intel Celeron CPU along with the Intel 810 chipset, plus 32MB SDRAM and a minimum 20GB hard disk. (See www.nokia.com/multimedia/mediaterminal.html.)

Nokia Mediaterminal

- ZapMedia ZapStation: ZapMedia's first product is a Linux-powered audio/video jukebox for serious consumer audiophiles. Beneath that stereo receiver look-and-feel lurks an 800MHz Intel Celeron equipped with 128MB RAM and a 30GB hard drive, running Linux kernel 2.4. The GUI is based on XFree86, Qt, Java and a customized Fresco Browser. (See www.zapmedia.com.)

ZapMedia ZapStation

- HP Digital Entertainment Center: HP's first product in the consumer entertainment device space has Linux 2.4 and the X Window System inside. Its embedded computer is based on a 566MHz Celeron, with 64MB RAM and a 40GB hard disk, and its interfaces include USB, Ethernet, HPNA, a built-in V.90 modem, stereo audio and video out for display on a TV. The device can download and record music and videos from the Internet and can read and copy music CDs onto its internal hard drive. (See products.hp-at-home.com.)

HP Digital Entertainment Center

- Bokks Portable Media Player: this versatile device connects to a broadband network via DSL, cable or optical fiber and plugs in to a TV to allow e-mail and internet access. It stores movies, music and other files on its internal multi-gigabyte hard drive, allowing them to be listened to or viewed whenever the user is near a TV, stereo or computer and is based on a 266MHz National Semiconductor SC1200 system-on-chip processor running Linux kernel 2.4.x. (See www.bokks.net.)

Bokks Portable Media Player

- Motorola DCT5000: Lineo announced that Motorola's Broadband Communications division had selected Lineo's Embedix for use in its interactive digital multimedia set-top appliances. The device includes features such as personal video recording, streaming media and home networking capabilities. (See www.linuxdevices.com/news/NS3134551333.)

Motorola DCT5000

- JCC iBOX-2: Japan Computer Corporation (JCC) announced the development of a new internet appliance based on National's Geode “set-top box on a chip”, running an embedded Linux operating system. The device contains 16-32MB DRAM, an 8MB DiskOnChip Flash disk and has a built-in Ethernet port. (See www.linuxdevices.com/news/NS4653311319.)

JCC iBOX-2

- VT Media VT-100: VT Media Technologies is partnering with Century Embedded Software to push into the low-cost broadband/IP set-top box market with a Linux-based solution. At CeBIT 2002, VT Media announced what they claim is the world's first broadband set-top box with a price point below $150 US. The device is based on National Semiconductor's

x86-based STB reference design along with Century's WebMedia software stack and Linux port. (See www.vtmt.com and www.censoft.com.)

VT Media VT-100

Server Devices

These devices are appliance-like “black boxes” used in home, small office and workgroup environments that are used as local servers (for local networking and file/printer sharing) or internet gateways (firewall, router, e-mail server, VPN, etc.). With a built-in Linux OS and open-source web server, these gadgets often provide point-and-click, browser-based setup and configuration by nontechnical users, and are unburdened by the high costs of proprietary software licenses. Here are a few embedded Linux-based examples:

- Filanet InterJak: the maker of this easy-to-use Linux-based appliance says it offers a Swiss Army Knife's worth of solutions to today's wired citizen. Inside, there's an x86 processor with 128MB of RAM, a multi-gigabyte hard disk, plus Linux and a host of other open-source software including Apache, MySQL and the Exim mail server. (See www.filanet.com.)

Filanet InterJak

- SnapGear VPN Routers: this family of compact, easy-to-configure, Linux-based devices offer gateway, router, firewall and VPN services. The low-end versions are based on Motorola ColdFire processors with 4-8MB of RAM and 1-2MB of Flash, while the high-end ones are powered by AMD SC520 processors with up to 128MB of RAM and 16MB of Flash. SnapGear makes extensive use of µClinux and is a contributor to the source tree. (See www.snapgear.com.)

SnapGear VPN Router

- Celestix Networks Aries: this compact appliance-like system is designed for easy-to-use resource sharing and internet access within small workgroups of up to 50 users. It makes use of laptop-style components and embedded Linux, resulting in reliable, fanless operation and a small enclosure footprint. (See www.celestix.com.)
- Sun Cobalt Cube: described as "a complete Internet and intranet server in a box", the Cobalt Cube combines server, gateway, router and VPN functions in an easy-to-configure, appliance-like blue box. The device is powered by embedded Linux running on an AMD K6-II CPU, equipped with up to 512MB of SDRAM and a high-capacity SCSI hard drive, plus interfaces for 10/100 Mbit Ethernet, serial console and USB. (See www.cobalt.com.)

Sun Cobalt Cube

- emWare DeviceGate: a highly compact yet fully functioning Linux-based gateway for specialized applications. Contains either a 486 or PowerPC processor, with 16MB system RAM and a 16MB DiskOnChip solid-state disk, plus interfaces for Ethernet and RS-232/485 serial. One example application of this device is remotely monitoring industrial pump controllers to help maintain water quality in water treatment plants. (See www.emware.com.)

Wireless Access Points

These small appliance-like devices provide 802.11 or Bluetooth wireless networking "hot spots" or "hot zones" in companies, shopping centers, libraries, downtown areas, airports, coffee shops and other public locations. Here are a few embedded Linux-based examples:

- Axis "Bluetooth Hot Spot": the model 9010 Access Point is based on the Axis ETRAX 32-bit RISC system-on-chip processor, equipped with 16MB

DRAM and 2MB Flash memory, a built-in Bluetooth transceiver and a 10/100 Mbit Ethernet interface. The device's embedded software consists of Axis' ETRAX µClinux port along with an Axis-developed Bluetooth Linux stack, plus a built-in web server for easy web-based installation and maintenance. (See www.axis.com/products/axis_9010.)

Axis Bluetooth Hotspot

- Ericsson's “BLIP”: the BLIP is a small, self-contained device that communicates wirelessly with mobile phones, PDAs and other kinds of mobile devices that are equipped with Bluetooth short-range wireless technology. Inside the BLIP is an ARM7TDMI-based 32-bit RISC processor with 2MB RAM and 2MB Flash, running µClinux, µClibc and Ericsson's Bluetooth Host Stack. The first version of the BLIP (BLIP C11) is no longer available and will be replaced by a new version (BLIP Net) later this year, according to a manager from Ericsson's Business Innovation group. (See www.linuxdevices.com/articles/AT9037120829.)
- Wireless Networks' BlueLAN: this device is built around NetSilicon's 32-bit ARM7TDMI RISC system-on-chip processor running µClinux. Its interfaces include a serial port, 10/100 Mbit Ethernet and a built-in Bluetooth V1.1 transceiver. (See www.wirelessnetworksinc.com/prod_bluelan.htm.)

Wireless Networks BlueLAN

- Musenki M-1/M-3: this recently announced family of access points is based on 200-333MHz Motorola MPC8241 system-on-chip processors, with up to 128MB SDRAM and 16MB Flash memory. The presence of internal Mini PCI slots offers a great deal of flexibility, such as the use of alternate versions of 802.11 or Bluetooth radios, as well as the addition of other desired functions and interfaces. Musenki took advantage of numerous open-source projects in creating their own unique embedded Linux implementation, including PenguinPPC kernel sources, PPCBoot, µClibc, BusyBox and more. (See www.musenki.com.)

Time Will Tell

Where and when will the greatest successes of embedded Linux emerge? What will be the highest volume products? Unfortunately, many killer app projects are kept under wraps until they're ready to hit the market, so we won't know about some of the most exciting ones until they make it to production. Watch this column for more news and perspective on this subject.

Acknowledgements

The following embedded Linux specialists contributed to this article: Lineo, Inc. (www.lineo.com), MontaVista Software (www.mvista.com) and TimeSys Corp. (www.timesys.com).

For the latest scoop (and details) on these and other devices with embedded Linux inside, visit LinuxDevices.com's “Embedded Linux 'Cool Devices' Quick Reference Guide” regularly (www.linuxdevices.com/articles/AT4936596231).

Rick Lehrbaum (rick@linuxdevices.com) created the LinuxDevices.com and DesktopLinux.com web sites. Rick has worked in the field of embedded systems since 1979. He cofounded Ampro Computers, founded the PC/104 Consortium and was instrumental in creating and launching the Embedded Linux Consortium.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

The Protocol Problem

Doc Searls

Issue #99, July 2002

Doc explains some of the reasons the Internet is so successful and yet limited.

Does the name Ungermann-Bass ring a bell? If it doesn't, chances are Sytek, Corvus, Bridge and Excelan are just as mysterious. Yet there was a time, back in the early- to mid-eighties, when these were big names in networking. If you read *Data Communications* back then, or *Communications Week*, you would have seen a lot of stories about (and advertising by) those companies, alongside more durable names like 3Com, IBM, Digital Equipment Corp. and Wang.

In those days the arguments were mostly about “pipes and protocols”. IBM pushed Token Ring (IEEE 802.5), which ran on special IBM wiring. DEC had a breed of Ethernet (802.3) that ran on its own coaxial cable. Corvus had Omninet, an Ethernet variant that ran on twisted pair. Wang had Wangnet, which ran on...something, I forget what. Same with AT&T's ISN (Information Systems Network). Everybody had their own arcane implementation of one “standard” datalink protocol or another. Each required their own special topology, of course, as well as their own transmission methods. You couldn't get away from articles comparing broadband against baseband and star vs. bus vs. ring topologies.

Around 1987 or so I did a bunch of work for Ungermann-Bass, which specialized in big networking jobs for big companies. U-B was naturally attracted to MAP, the Manufacturing Automation Protocol developed by General Motors in 1982 to standardize networking within the company's many factories and manufacturing facilities. MAP came to be known as MAP/TOP after it was mooshed together with Boeing's Technical Office Protocol. MAP/TOP was conceived around IEEE 802.4, which required a bus topology, although it distributed data with a token in the manner of IBM's Token Ring networks. This was supposed to be a more robust and noise-immune approach than the more popular Ethernet alternatives.

U-B's efforts were well meaning and highly regarded by the press but were undermined by Ethernet's success, even on factory floors. Not only was Ethernet relatively cheap, but it integrated nicely with networked PCs, which by default used Ethernet as well—thanks to the successful efforts of companies like 3Com, which made Ethernet interface cards by the millions.

All due credit to U-B, there was no way to know MAP wouldn't catch on, or even that the combined industrial heft of General Motors and Boeing couldn't lever the market to 802.4. Even though they cost hundreds of dollars apiece, 3Com's Ethernet cards were cheap and ubiquitous. As for higher-level transport and internetworking protocols, there was even less reason to believe that TCP/IP would ultimately define the most universal network ever created: the Internet.

What's amazing to me now, looking back over the years, is how long it takes to establish a given protocol, and how so many huge companies bet and lost their ranches on protocols that markets never ratified.

Today we're accustomed to thinking of “internet time” as something that moves very fast. We credit Moore's Law with improving new computers while obsoleting old ones in less time than it takes to grow a crop of asparagus. Yet protocols mature in about the time it takes an apple seed to become a fruit-bearing tree. Craig Burton, CEO of JanusLogix, says,

> The time it takes to invent, create, develop and standardize on a protocol for any given service is way out of alignment with what is happening in the marketplace. Take LDAP. LDAP v3 was introduced some six years ago at this point. You think the Linux kernel takes a long time to move, it is lightning fast compared to how fast a protocol moves.

I just checked and it's now closer to seven years. The same thing seems to be happening, by the way, with IPv6. Any bets on how long it's going to take before it becomes a de facto standard?

So the protocol problem is a big reason why the Internet is so successful yet so limited.

Look at the core protocols that enable the Net's basic services: HTTP, FTP, LDAP, SMTP, POP3, IMAP and SNMP, to name seven. In effect, these are the Net's true infrastructure because they govern the few services the Net can universally support. The Net grew around this set of protocols precisely because they were simple, unencumbered by corporate ownership and easily ubiquitized. The X.400 standard defined a rich set of e-mail functionalities, but it was a pain in the butt to implement. SMTP and POP3 were free and easy, so those protocols came to serve as the universal infrastructure for e-mail service.

Yet even today no internet protocols support the rich and complex file, print and directory services that come standard with, say, Novell's NetWare.

Many internet services remain undeployed (or underdeployed) for lack of enabling protocols. Instant messaging (IM) is one example. The fact that any of us can download an IM client from AOL or Microsoft doesn't mean that the Net itself possesses any kind of IM standard. AOL, Microsoft, Yahoo! and Lotus all have IM systems that use arcane, proprietary and closed protocols, each restricting their own clients to their own servers. Thus, AOL's AIM clients are like browsers that can only visit one web site. Without the IM equivalent of an HTTP or an SMTP, there is no IM equivalent of Apache or Sendmail. The only serious candidate at this point is Jabber's protocol (www.jabber.org), but it's still new—more shrub than seed, but far from a fruit-bearing tree. Meanwhile, the Net has no IM infrastructure.

Security is another one. Computing has been around for 50 years or more, and we still think about security largely in terms of firewalls. (I got an e-mail this morning from a guy who can't see one of our web pages here at *Linux Journal* because his Fortune 500 company's firewall excludes pages that contain the word “suck”.)

Firewalls are a medieval solution to a 21st-century problem. Their models are forts and castles, moats and drawbridges. The result is what Craig Burton calls

> Web noir—a technological Dark Age obscured by the apparent brilliance of the Internet, as we know it. The dark—that noir—is what we don't see, what we don't know because it doesn't yet exist.

(See www.craigburton.com/stories/storyReader$19.)

The first bright hope for a renaissance, Burton says, is XML (eXtensible Markup Language). With the recent development of SOAP, WSDL, WSIL, UDDI and WSIL, XML makes web services possible. Some of these services look so promising that Microsoft, IBM and Sun are pushing their own web service frameworks: .NET, J2EE and SunOne, to name three.

Yet for all their inclusivities (especially in promotional rhetoric) frameworks have an exclusive effect. They lock in customers and lock out competitors. If the Net has proven anything, it is that exclusive architectures, frameworks and environments all fail as universal infrastructure. Yet they have their appeal, even to people who ought to know better. Take Mark Forman, the White House technology czar. In April 2002, the *Seattle Times* reported that Forman was operating on the erroneous assumption that Microsoft's Passport technology might serve as a kind of national identity infrastructure.

The *Seattle Times* wrote,

> Forget about a national ID card. Instead, the federal government might use Microsoft's Passport technology to verify the on-line identity of America's citizens, federal employees and businesses, according to the White House technology czar.
>
> On Sept. 30, the government plans to begin testing web sites where businesses can pay taxes and citizens can learn about benefits and social services. It's also exploring how to verify the identity of users so the sites can share private information.

(See seattletimes.nwsource.com/html/businesstechnology/134438173_passport18.html.)

Passport will fail that mission for the same reason AOL's and Microsoft's instant-messaging systems will ultimately fail as universal standards, no matter how popular they are today: they are owned by people with vested interests in keeping their properties closed and exclusive.

Identity should be another of those fundamental web services. Your self, your car and a drill press on a factory floor should all have unique identities that make sense in the universal context of the Net—someday. But how will that happen? Will it depend on yet another protocol or some new set of protocols? By what decade?

So let's raise a question: if we can't speed the evolution of protocols, can we at least find a way to reduce or eliminate their role as a gating factor for progress? Can we eliminate this drag on Moore's Law, as well as our own sense of possibilities?

When we find the answer, Burton says, “the result will be as transformative as the discovery of calculus, which accelerated scientific revolutions and helped bring on the Renaissance.”

Doc Searls is senior editor of *Linux Journal*.

Archive Index Issue Table of Contents

Advanced search

License Defamation

Lawrence Rosen

Issue #99, July 2002

How the GPL has been sorely defamed.

Defamation is the “communication to third parties of false statements about a person that injure the reputation of, or deter others from associating with, that person.”

Unfortunately, under the law, you can't be sued for defaming a software license. If I could, I'd sue Microsoft for what it has said about the GPL. Their statements are false and are obviously intended to deter companies from associating with GPL-licensed software—specifically Linux.

The latest onslaught from Redmond came in the “Royalty-Free CIFS Technical Reference License Agreement”, under which Microsoft offers to license its CIFS technical specifications, along with certain of its patents necessary to implement those specifications, on a royalty-free basis. (Read it yourself at msdn.microsoft.com/library/default.asp?url=/library/en-us/dnkerb/html/Finalcifs_LicenseAgrmnt_032802.asp.)

At first I was thrilled to see that Microsoft was willing to license its patents on a royalty-free basis. Members of the Open Source community have long insisted on royalty-free patent licenses for any technology needed for interworking or other standard implementations. But there are so many exclusions and limitations to the CIFS license that this license is useless for anything except Microsoft's propaganda purposes; I can't imagine anyone who will want the CIFS technical specifications under those terms.

For example, the license is limited to implementations on “Non-Microsoft Platforms”. Whoopee! That's how Microsoft stops direct competition at the doorstep so that nobody can write better software to replace any of Microsoft's own software.

It was the insult in the CIFS license that really caught my eye. The CIFS technical specifications and patents are not available for use with software that is distributed under an “IPR Impairing License”. This term is defined as:

> ...the GNU General Public License, the GNU Lesser/ Library General Public License, and any license that requires in any instance that other software distributed with software subject to such license (a) be disclosed and distributed in source code form; (b) be licensed for purposes of making derivative works; or (c) be redistributable at no charge.

So Microsoft again reveals that the only real competition it fears is from Linux and other open-source software.

The term IPR Impairing stands for intellectual property rights impairing. In Microsoft's confused lexicon, the GPL impairs intellectual property rights. Whose? How? Is the impairment good or bad?

There is nothing in the GPL that impairs anyone's intellectual property rights, and it is outrageous for Microsoft to assert otherwise. Anyone can use GPL-licensed software without any effect at all on his or her own intellectual property. For example, you don't have to publish your source code merely by running your software under Linux. You can even modify GPL-licensed code—including Linux—and as long as you don't distribute your modifications you incur no obligation whatsoever to distribute the source code to your modifications. (By way of contrast, don't you dare to reverse-engineer, much less modify, Microsoft code!)

You are not forced to license your software under the GPL and to distribute your source code—except in one rare situation: if you take GPL-licensed software, modify it and distribute your modified version, you must license your modified software under the GPL.

This reciprocity provision is based upon a legitimate bargain. In essence, the licensor is giving you broad rights to his or her software, including the right to create and distribute modifications, but only if you agree to reciprocate by returning your distributed modifications to the commons. You must pay for the right to create and distribute modified software, but the currency for your payment is not money—it is your improved software itself.

How does that differ in principle from the obligation to pay the asking price for Microsoft software?

The owner of GPL-licensed intellectual property is establishing a price for a license to his or her property. You are always free to decide not to make and

distribute modifications to GPL-licensed software, but if you elect to do so, you must pay the price set by the owner of the original software. You received a benefit from the free software you modified to create your own software, and the price for that benefit is reciprocity.

That is simply capitalism at its finest. The GPL is not, despite Bill Gates' protestations to the contrary, the end of capitalism as we know it.

What can be far more serious challenges to capitalism are the monopolistic practices that Microsoft continues to engage in. That company pretends it is offering a royalty-free license to CIFS, but the license doesn't apply to implementations on either the Microsoft platform or on the major challenger to that platform, Linux.

Welcome to Microsoft's white elephant sale. I suggest you don't buy.

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

email: lrosen@rosenlaw.com

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

Archive Index Issue Table of Contents

Advanced search

Letters

Various

Issue #99, July 2002

Readers sound off.

Looking for a Fix

Mick—thanks for your Paranoid Penguin columns; I always open my new issue of *LJ* right to it to see what you've got in store each month. The “Hardening Sendmail” article [*LJ* April 2002] was another winner. I just wanted to point something out regarding your recommendation not to run a sendmail dæmon if only sending mail. I used to do just that but found something unfortunate: if a send attempt fails, then sendmail could queue it for later delivery. If the sendmail dæmon wasn't running, well, “later” would never come. I wouldn't know about the failed delivery (especially without the usual informative messages from sendmail after four hours, and after five days). So, I run sendmail (postfix, actually). If you've got a better solution, I'd love to hear it. And I'm sure other readers would, as well.

—David

Mick replies: Thanks for the kind words! You're right, if sendmail isn't running as a dæmon, queued mail will remain queued indefinitely. I should have mentioned that common practice is to set up sendmail to be run from cron periodically with the -q flag. For example, this sample crontab line (adapted from one in Olaf Kirch's Linux Network Administrator's Guide) invokes sendmail in “flush queue” mode every 15 minutes:

```
0,15,30,45 * * * * /usr/sbin/sendmail -q
```

Obviously, that has to be in the crontab of an account authorized to run sendmail this way—usually root. And depending on how much outbound mail you deal in, you may not need to run sendmail every 15 minutes—hourly may

*suffice. Postfix most definitely rocks, by the way. And my friends who use qmail *really* like qmail.*

P3P Complexities

I was surprised to see your article about P3P in the April 2002 issue of *Linux Journal*. Since 1999, P3P has been shown to be orthagonal to privacy, despite claims made to the contrary. I don't really see that the W3C has done a lot to allay fears of P3P. P3P is overly complicated and is geared toward collection of user information, not protection of it. The only serious difference between P3P and non-P3P sites is convenience in giving away your personal information. It's not even a legal help: just because a company does something illegal does not mean the average Joe can do anything about it.

The protocol could be greatly simplified and need not have any information about the user at all. Also, even if the protocol forces contact information to be given to the user, there is no easy method for the browser to determine its validity, and it is no guarantee the company will listen. In other words, “same ole same ole”, but more complex. A protocol that really is designed to protect users' privacy will never need to know anything about users except their privacy preferences. There is absolutely no need for other information, yet P3P includes a large amount of detailed and personal user information. You have to ask yourself why this has been made so complex and so heavily geared toward data acquisition.

—Anonymous

Larry replies: I have already received comments to the effect that P3P doesn't ensure privacy. Certainly nothing I said was meant to indicate that it is a perfect solution to the problem. I just thought then, and still do, that automated privacy protections are far more protecting of the average internet user than the “click here to read our privacy policy” that is so typical nowadays. Whether people will use even the automated tools, or whether companies will honestly comply with their promises, are open questions. I do know that if a web site promises me privacy through its, and my, P3P settings, and it subsequently releases private information about me contrary to my express preferences, I'll sue. And if an average-Joe user comes to me with a situation like that, I'd consider handling the case as a class action lawsuit and demand attorney's fees.

Minimum Distro?

I have been a Linux user since Red Hat 4.2 and picked up my first *LJ* about two years ago. I have always poked about in all the different directories and have always wondered “What is really required here?” I mean, what is the minimum requirement for a working Linux distribution, without any user apps, just

something that loads, prompts for a login, then takes you to a shell prompt and lets you log off or shut down. No lynx, no elm, no sendmail, no anything. I have always thought it would be a great personal educational undertaking for myself to attempt to create my own distribution...but I've no idea where to begin.

—Blake Tullysmith

Yes, you can build a tiny distribution with just a kernel and a shell. You might want to start with Brian Finley's “Brian's Own Embedded Linux” and remove software from it. BOEL fits on a floppy, and two articles about it appear on embedded.linuxjournal.com. See www.linuxdevices.com/articles/AT9049109449.html and www.linuxdevices.com/articles/AT5974781081.html.

—Editor

Open vs. Free

A note in the UpFront section of the May 2002 issue describes the Fórum Internacional do Software Livre as an “open-source event” and says that the state of Rio Grande do Sul promotes and leads in use of “open source”. Actually the event is about free software (software *livre*, in Portuguese), and the state's policy is to promote free software.

I launched the Free Software movement in 1984 to campaign for computer users' freedom to have a community. To make freedom and community possible, we began developing the free software GNU operating system, which is the basis of the GNU/Linux system that your magazine is dedicated to. Our idealism made our community possible.

The Open Source movement has contributed to our community, and its supporters have a right to promote its views, but we are not part of them. The Porto Alegre event's organizers and the government of Rio Grande do Sul know the difference between the two movements, and they chose the Portuguese term for “free software” to say where they stand. I ask that you not label Free Software movement supporters, our work, our events or the community we built, with the term of the other movement. For more explanation of the difference, see www.gnu.org/philosophy/free-software-for-freedom.html.

One other correction is that I will not be speaking at this event. It was originally planned that I would, but I visited Porto Alegre for the Fórum Social Mundial in January 2002, so we decided it would be more interesting for Bob Chassell, cofounder of the Free Software Foundation in 1985, to go this May instead of me.

—Richard StallmanPresident, Free Software FoundationChief GNUisance of the GNU Project

Porn Correction

In his article “Intrusion Detection Systems”, Pedro Bueno proposes a sample Snort rule that is supposed to alert us of “any porn web access attempt” from the private network. Unfortunately, in its current form, the rule will trigger only when a user is *posting* data containing the string “free porn” since it only filters outgoing traffic. In order to detect actual access attempts, the rule should filter the incoming traffic:

```
alert tcp $EXTERNAL_NET  80 -> $INTERNAL_NET any
  (msg: "Web Porn Access Attempt";
   content:"Free porn";
   nocase; flags:A+);
```

An excellent article, though.

—Florin Malita

Errata

On page 14 of the May 2002 issue, in the “It's Trivial” section, question seven, the title of William Gibson's novel should read *Neuromancer* rather than *Necromancer*. Sincere thanks to the many Gibson fans for pointing out this unfortunate typo.

Due to a printer error, page 20 of the June 2002 issue appears twice, leaving out page 21. Please see the full article on our web site: www.linuxjournal.com/article/5839.

Archive Index Issue Table of Contents

Advanced search

UPFRONT

Various

Issue #99, July 2002

Beer Contest, II

Thanks to everyone who decided that looking for a miniscule beer bottle among the pages of *Linux Journal* for a chance at some auto accoutrements is a worthy way to spend what little free time you have. Jon maddog Hall even sent in a correct answer (albeit too late to qualify for the prize), telling us “You can't hide beer from maddog”, as if we didn't know.

Many hundreds of readers were sharp enough to find the beer bottle on the table on page 93 in the “Crystal Space” article. Some even picked up on the hint by way of the 3-D glasses suggestion. Others should spend more time looking for beer (or better yet—a job) and less time drinking it, for they “found” beer in the most remarkable places in the magazine's pages. For these folks we're offering another chance. To make the new start fresh, this time you'll be looking for something completely different—a mug of beer—and no, it's not the one pictured here. This one is an *example*. For this month's prize we'll be letting you help us clean out our warehouse by accepting some pristine vintage *Linux Journal* XL T-shirts. This time, correct responders numbered 200-300 will win.

Send your mailing address with your answer to info@linuxjournal.com. Good luck!

The Game too Addictive for Debian?

If you're doing without *Tetris* or *Solitaire* one day at a time, stop reading right now. The free SDL-based game *Frozen Bubble*, in which you control a cute penguin and lob colored bubbles into a geometric pattern while a chill-room soundtrack plays, is so addictive that user Nick Moffitt reported it to the Debian bug tracking system as a “Title 1 Controlled Substance”.

Bug Report: bugs.debian.org/cgi-bin/bugreport.cgi?bug=143176

Game Home Page: www.frozen-bubble.org

—Don Marti

LJ Index—July 2002

1. Estimated millions of dollars saved annually by Largo, Florida by running Linux applications on Linux servers and thin clients: 1
2. Number of distinct domains that lead to the Tina's Webcam site: 4,525
3. Millions of dollars Bertlesmann loaned Napster, through April 2002: 85
4. Millions of dollars more Bertlesmann would be willing to pay Napster, as of April 2002: 15-30
5. Billions of dollars for which Napster was originally sued by the RIAA on behalf of Bertlesmann and other companies: 20
6. Debian Linux CDs burned by the government of Extremadura, a rural area called the poorest in Spain, for distribution to schools through newspaper inserts: 80,000
7. Number of schools in Extremadura: 670
8. Number of technology centers in Extremadura: 32
9. Expected yearly savings to the Extremadura community in millions of dollars: 7
10. Number of government offices and schools expected in the Extremadura extranet by the end of 2002: 1,478
11. Number of Extremadura's teachers expected to be trained on the use of Linux in the classroom: 15,000
12. Millions of dollars spent by MCA Records to make and market the album *Ultimate High* by Carly Hennesey: 2.2
13. Copies of *Ultimate High* sold as a result of MCA Records' efforts: 378
14. Percentage of records that become profitable, according to record industry sources: 5
15. Average number of records that must be sold for a major label release to break even: 500,000
16. Size in inches at its widest side of the Transmeta Crusoe-based OQO “modular computer”: 4.9
17. Weight in ounces of the OQO: 9
18. Size in GB of the OQO's hard drive: 10
19. Expected battery life in hours of the OQO: 8
20. Trillions of calculations per second of the Linux-powered supercomputer HP sold to the US Dept. of Energy in April 2002: 8.3

21. Number of Intel Itanium 64-bit processors in the Energy Dept.'s new Linux supercomputer: 1,400
22. Millions of dollars the Energy Dept. will pay for its new supercomputer: 24.5

Sources

1: ZDNet

2: Ben Edelman, Harvard Law case study

3: *New York Times*

4-5: ABC News, MP3 Newswire

6-11: Wired News

12-15: *Wall Street Journal*

16-19: LinuxDevices.com

20-22: CNET

MIT Scheme Offers Historic Strength, Modern Convenience

I have a very real need for programs to treat other programs as data because I do a lot of work in program-writing programs, or “metaprogramming”, for CAD applications. I've done this sort of programming in Python and in C++, but I'm increasingly finding the Lisp family of languages to be the power tool I have always wanted. Recently, I was very pleasantly surprised with the power and features of MIT Scheme Release 7.7.0, released in March 2002.

Scheme is now more than 20 years old but is very robust, containing numerous concepts only now being seen in late-modern languages, such as C++ and Java. For instance, Scheme has continuations, where the state of a computation becomes in essence a controllable object. This powerful and general facility is useful for concurrency, modeling, multithreading and arbitrary control flows.

MIT Scheme's library, SLIB, contains support for working with getopt, HTML, relational databases and other functionality similar to the Python or Perl libraries. There is a very powerful Scheme Object System OOP library, which has features similar to the Common Lisp Object System (CLOS).

Scheme seems to have a close kinship with Common Lisp. However, MIT Scheme is not only GPLed, but feels more approachable, with a simpler syntax

than Common Lisp. For my work, Scheme's scoping model and use of namespace are somewhat better than Common Lisp programs.

To get started with MIT Scheme, see the project home page: www.swiss.ai.mit.edu/projects/scheme. Scheme resources are available at www.schemers.org. See also *The Scheme Programming Language: ANSI Scheme*, Second Edition, by R. Kent Dybvig, ISBN 0-13-454646-6.

—Michael Baxter

It's Trivial

Questions

Q1 What company started in 1938 from a garage at 367 Addison Avenue, Palo Alto, California with a resistance capacity audio oscillator as its first product?

Q2 It could have been: Calex, Elcal, Calecom, Elcom, Calcomp, Digicom, Tronicom, Comptek, Computek, Esscotek or Ectek. What name was finally chosen for this startup?

Q3 Incorporated as Computing-Tabulating-Recording company in 1911, it formally changed its name in 1924. What is the company called today?

Q4 The story goes that the campus of the University of California, Berkeley had a machine, called Ingres, connected to the ARPAnet and another machine, called Ernie Covax, that was home to the Berkeley UNIX Project, which was connected to a network known as BerkNet. These machines were interconnected, but there was no way of moving mail from one network to another. Eric Allman wrote a software program to transfer mail between these two networks and later went on to found a company. Which?

Q5 Many talented programmers, unhappy with what was going on at Fairchild Semiconductors left to create their own startups. Intel and AMD are just two examples of semiconductor companies that arose out of Fairchild Semiconductors. What did the press dub these startups that arose from Fairchild?

Answers

A1 Hewlett-Packard

A2 Intel

A3 IBM

A4 Sendmail, Inc.

A5 Fairchildren

—Sumit Dhar

And How Many Times Will the Kernel Be Rewritten before This Is Finished?

By the time you read this, Free Radio Linux (radioqualia.va.com.au/freeradiolinux) will be about a quarter of its way through a reading of the Linux kernel by a digitized voice. The voice is a creation of r a d i o q u a l i a (radioqualia.va.com.au), an "on-line radio station aiming to open an electronic portal into the eccentricities of antipodean radio space". The site explains,

> r a d i o q u a l i a is engaged in the exploration of sound and media within the context of philosophical speculation. Informed by the discrete discourses surrounding science, art and philosophy, r a d i o q u a l i a is attempting to simulate the introspectible and seemingly monadic properties of sense-datum, through sound.

Since Free Radio Linux's kernel (they don't say which version) contains 4,141,432 lines of code, the reading is expected to take 14,253.43 hours, or 593.89 days. Transmission began on February 3, 2002, the fourth anniversary of the Open Source Initiative. You can listen in through Ogg Vorbis.

—Doc Searls

Stop the Presses: Hollywood Turns Up the Heat

It was law professor and author Lawrence Lessig who first described the battle for control of the Net as a kind of civil war between Northern and Southern California: between Silicon Valley and Hollywood.

The North, he says, "believes in a free exchange of ideas". The modern South wants to protect intellectual property the way the old South wanted to protect its privileged plantation system. If you want to traffic in any of its copyrighted material, you have to visit "their plantation and seek permission from the master". And, "if you develop technology that interferes with their right of perfect control, you will be punished."

In this civil war, everyone who wants to protect the Net's wide-open frontiers is a Siliconian, and everyone who wants to maintain absolute control over the distribution and use of "content" is a Hollywoodian.

The latter would include Jamie Kellner, the chairman and CEO of Turner Broadcasting, who was installed by AOL's Bob Pittman in the wake of the AOL/ Time Warner merger. In an April 2002 interview with Cableworld, Kellner said, “I'm a big believer we have to make television more convenient or we will drive the penetration of PVRs and things like that, which I'm not sure is good for the cable industry or the broadcast industry or the networks.”

When asked to explain why, he continued:

> Because of the ad skips....It's theft. Your contract with the network when you get the show is you're going to watch the spots. Otherwise you couldn't get the show on an ad-supported basis. Any time you skip a commercial or watch the button you're actually stealing the programming.

PVRs are Personal Video Recorders. The most popular PVR is the Linux-powered TiVo. Its main competition comes from ReplayTV, which sells under the Panasonic and SonicBlue brand names. ReplayTV PVRs have the power not only to record TV programs, but to share them over the Net. To Hollywood, sharing is theft. In testimony before Congress, Jack Valenti, chairman and CEO of the Motion Picture Association of America (MPAA), said, “The potential undoing of America's greatest export trade prize...is the theft of movies in both the analog and digital formats.” In that same testimony, he outlined a three-prong attack on “piracy” that began this way:

> First, we have taken on the task of protecting copyright laws in the courts....We have to insist that copyright laws cannot be casually regarded, for if those laws are shrunk or loosened, the entire fabric of costly creative works is in deep trouble.

So, as I write this (on May 2, 2002), ReplayTV finds itself the subject of a lawsuit brought by a raft of TV and film studios. In the course of discovery, the plaintiffs filed a motion asking ReplayTV to turn over precise customer usage records. Cory Doctorow, outreach director for the Electronic Frontier Foundation, describes what happened next:

> As we understand it, the Court ordered ReplayTV to change its software to include spyware that will capture its customers' clickstream—every commercial skipped, every show watched, every recording shared over the network. The plaintiffs—a cabal of Hollywood studios and TV networks—asked for this information as part of the pretrial discovery process, and when ReplayTV responded that it did not have the data or the means to collect it, the judge ordered them to change the software to collect the information within 60 days. When ReplayTV tried to safeguard its

> customers privacy by making the mandated spyware an opt-in process, the judge denied the request.

By the way, TiVo does not collect individual data—only aggregate statistics—and all data gathering is opt-out for customers. But the precedent here puts TiVo no less at risk than ReplayTV. Valenti's legal attack strategy is sure to send a message to the legal departments of the world's TiVos.

In a recent interview with *Business 2.0*, Lessig explained the strategic context:

> The problem today is that the words "intellectual property" have become captured by people like my friend Jack Valenti, who goes around talking about intellectual property not as a balance but as an extreme; not as something that we're supposed to be constantly restriking as technologies change to make sure it doesn't stifle innovation, but as a tool that the dinosaurs can use to make sure there are no mammals in the future.

Cory Doctorow puts it another way, "The role of the technology industry is to blaze new trails that create new opportunities for Hollywood. The role of Hollywood is to seek injunctive relief from those opportunities."

But Silicon Valley has some friends whose jobs cross the boundaries. Mark Cuban, most familiar these days as the high-profile owner of the Dallas Mavericks NBA basketball team, is also chairman of HDNet, the first all-HD (high definition) TV network. At a speech to the National Association of Broadcasters, Cuban called on the broadcasting industry to "just completely ignore" Hollywood in the fight over copyright and intellectual property, calling the litigious studios a "chicken little environment".

—Doc Searls

They Said It

I would point out that linked lists, mark-and-copy garbage collection, and the Tab key are all patented too. Somebody who always carefully checked first for software patents would never write anything at all.

—Martin Pool, in the rsync FAQ

VANTEC does offer a limited warranty but as a practical matter that warranty is seldom applicable to competitive and particularly destructive robot contests.

—Vantec.com web site

The public interest in public-domain intellectual property freezes dead with the humble birth of a cartoon mouse on a tabletop in Kansas City. The Mouse is flash-frozen in legal ice. He's unrotting. He's undying. He's cryogenically preserved....In ancient Rome, folks thought it was pretty decadent when the Emperor Caligula made his horse a Senator. But in the modern US Senate, there's a Senator who's a cartoon mouse!

—Bruce Sterling

Don't be fooled. The BPDG standard is not about stopping “piracy”. It's about Hollywood regaining some measure of control over what you can and can't do with television. It's about cramming the VCR genie back in the bottle and giving Hollywood the power to bring new technologies to heel before they can deliver new capabilities to consumers.

—Fred von Lohmann

If you love wealth more than liberty, the tranquility of servitude better than the animating contest of freedom, depart from us in peace. We ask not your counsel nor your arms. Crouch down and lick the hand that feeds you. May your chains rest lightly upon you and may posterity forget that you were our countrymen.

—Samuel Adams

We're moving towards a “Creole” of technological concepts. The idea comes from language theory, specifically Steven Pinker's work where adults come together in an area with lots of different languages and end up coming up with a broken, lumpy language that is put together as a pidgin language. When the next generation comes along, however, it becomes more sophisticated and develops into a real language, then called a Creole. You only have to watch kids today using technology to realize the similarities, and that we adults are very much the pidgins.

—Douglas Adams

Archive Index Issue Table of Contents

Advanced search

From the Editor

Richard Vernon

Issue #99, July 2002

In his column this month, Rick Lehrbaum consolidates the embedded Linux arena into three basic categories: telecommunications infrastructure, handheld mobile devices and internet-edge devices (this includes server/gateway/firewall-type devices as well as digital set-top boxes). With the exception of the desktop, this includes just about everywhere one might expect to find Linux.

Rick has been tracking the progress of Linux in these areas for the last couple of years, reporting on tradeshows and devices that, exceptionally, ran Linux. However, that has changed very quickly, and now the trend that is impossible to ignore in Rick's columns is that the computing world is increasingly embedded, and embedded is increasingly Linux.

As Doc Searls has been saying often of late, “Linux has won.” Consequently, the success stories that were so compelling two or three years ago about some company switching their servers to Linux have long since ceased to be news. This doesn't mean that there aren't fights to be fought and progress to be made, but it does mean that Linux and the open-source development model have certainly been proven as viable for every level of computing.

An indication of the success of Linux was the Open Source Development Lab's announcement at LinuxWorld, New York earlier this year of the Carrier-Grade Linux Working Group initiative—also discussed by Rick (*LJ*, May 2002). The group's goal is to aid in the creation of any component necessary to make Linux the best choice for carrier-grade applications.

In a related event, Ibrahim Haddad, whom many readers will recognize as a fairly regular contributor to *LJ*, e-mailed me recently to let me know that, after three years of hard labor, Ericsson is taking some of his group's work to the Open Source community. Ibrahim works as a researcher at the Ericsson Corporate Unit of Research in Montréal, Canada. The most important part of the work that is being open-sourced is an architecture for security on telecom-

grade Linux clusters. It is called DSI (Distributed Security Infrastructure). Ericsson will be sponsoring its development and is looking to the Open Source community for help. They have officially joined the OSDL to work with other members of the Carrier-Grade Linux Working Group.

The DSI team has authored an article in this issue explaining the project (see page 92). The project's web site is not yet up, but anyone interested in contributing to it may contact any of the team members listed at the end of the article.

Richard Vernon is editor in chief of *Linux Journal*.

Archive Index Issue Table of Contents

Advanced search

Best of Tech Support

Various

Issue #99, July 2002

Our experts answer your technical questions.

I Have No "core" and I Must Debug

The default action for a dozen UNIX signals like SIGSEGV is to abort and generate a core file as a dump of the memory image of the faulty process. On Red Hat, when getting such signals, the processes abort but do not generate any core file. The only way to know a faulty process was sent such signals is to strace it. If it is possible, where on Red Hat is the core generation configured?

—Pedro Guedes, pmg01@netc.pt

The ulimit command controls the size limit of several things in your Linux system, among them, the size of the core files. Issuing a **ulimit -c 0** will tell Linux to disable core files (since they will be of zero size), **ulimit -c *xxxx*** will tell Linux to allow core files of up to xxxx blocks in size. **ulimit** may be set in /etc/profile for all users or in your home directory's .bash_profile. Typing **ulimit -h** gives help on this command. If you edit the file /etc/security/limits.conf, there you may find a line something like:

```
*  soft   core  0
```

or:

```
*  hard   core  0
```

That last 0 on each of these examples indicates that the core file size should be no bigger than 0KB; therefore there are no core files generated. Hard and soft refer to the kind of limit imposed, "hard" being more strict. Change that 0 to something else, depending on the biggest core dump file that you may expect. This file (/etc/security/limits.conf) is part of the PAM (Pluggable Authentication Modules) system, which most distributions, including Red Hat, use.

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

X on an Old SiS Video Card

I have installed Red Hat 7.2 on an old (1994-ish) colussus, but I cannot get X to run as it doesn't see the monitor, which is a CTX 1565CD. However, when I try to run Xconfigurator or any other X config tool, it does not pass the X config test. When started, the PCI probe returns the entry Silicon Integrated Systems (SiS) SG86C20, which does not appear anywhere on the list of supported cards.

—Jim Logan, jim_message@hotmail.com

The Linux Hardware Compatibility Guide (www.ssc.com/mirrors/LDP/HOWTO/Hardware-HOWTO) says you can use XF86_SVGA in a 3.x version of XFree86, or the “sis” driver in a 4.x version. The SVGA driver is a good option to try because it works with almost any card, although it will not be accelerated. Also, if you have a newer version of X, see if your card supports VESA by enabling it in the kernel and using the “vesa” driver for X.

—Chad Robinson, crobinson@rfgonline.com

What's My IP Address Today?

I have Cox.net broadband service with an SMC router yielding a DHCP address for my home network. On the network, a Linux box is the virtual server. What I would like to do is poll the router to find the LAN side address and e-mail it to myself. The purpose of course is to be able to remotely connect using the DHCP IP address. The web page configuration on the router always starts at the login page regardless of the URL used (security). SMC does not seem willing to respond at all to my question.

—Tom Mautner, mautner@cox.net

Have you considered a dynamic DNS service? That may let you do an end run around to collect the address internally. A service such as ZoneEdit will allow you to update your address automatically via a web query, which you can automate with lynx.

—Chad Robinson, crobinson@rfgonline.com

Send yourself an empty e-mail, and you will see that the Received header lines will contain the IP that your router used when it connected to Cox's mail relay. You'll see something like this:

```
Received: from manyroads.ssc.com (manyroads.ssc.com
        [192.168.3.58]) by mail.ssc.com (Postfix) with ESMTP
```

and you can get the IP from there. Not high tech, but it works.

—Marc Merlin, marc@merlins.org

If the router is doing NAT, and you want the address of its internal interface, run:

```
route -n | mail you@example.net
```

—Don Marti, info@linuxjournal.com

X Takes over Both Video Cards

I have two graphic cards in my box, one is ATI Rage XL (PCI) and one is ATI RADEON VE QY (AGP). I would like to control the AGP card from a kernel module we have written (which displays on a projector), and I would like to let X control the PCI card (which displays on a flat panel monitor). The problem begins when we use startx. The PCI controlled monitor starts X correctly, but our kernel module no longer has control of our AGP card (no image displays on the projector). Does anyone know why X seems to be taking away our control of the AGP card?

—Jing Xu, jing@cs.unm.edu

Try forcing X to use a specific bus ID using the "BusID" parameter. Perhaps X is trying to use both cards as a multihead configuration, but you should be able to prevent it from doing that.

—Chad Robinson, crobinson@rfgonline.com

Fixing ext2 Filesystem Damage

Question: after a power blackout I am having trouble with my drive. I am using the ext2 filesystem. With the help of the tomsrtbt floppy distribution and a couple of documents from the LDP, I have been able to boot in single-user mode. But when I try to execute the network script, the system sticks after echoing the following:

```
Disabling IPv$ packet forwarding
sysctl: ip forwarding off
```

What can I try next?

—Roberto Kruse, rkruse@ieee.org

I would advise you to boot from a rescue disk and run an e2fsck on the partitions of your damaged hard disk for an automatic check:

```
e2fsck -p -v /dev/<partition_to_fix>
```

Keep in mind that messing around with your disk, even with fsck and e2fsck, may result in loss of data, so be careful that you understand what you are about to do.

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

You can try booting Linux with **linux single** or **linux init=/bin/bash** if all else fails. Type this at the LILO prompt. After that, you can look at the output of **rpm -Va**, which should tell you which packages have modified or missing files, and you can then re-install the said packages (with **rpm -U --force** ***/location/of/package.rpm***) All that said, it's probably as good a time as any to upgrade your system to the latest Red Hat version, which should also fix your problem in the process.

—Marc Merlin, marc@merlins.org

How to Mount a USB Device?

I would like to access a Disgo 16MB USB Flash RAM device from Mandrake Linux. Will Linux automatically mount this device? If so, how can I find it?

—Greg, gregory_connor@yahoo.co.uk

I believe that Mandrake uses the usbdevfs pseudo-filesystem, though I don't know where it mounts. Type **mount** to see. My Red Hat machine mounts it on /proc/bus/usb.

—Ben Ford, ben@kalifornia.com

Linux supports USB storage devices if you have a recent kernel version. According to Mandrake's web site, they shipped 8.1 with kernel version 2.4.8. However, because the device drivers have evolved quite a bit in the last few months, it may be helpful to update to the latest 2.4.x version, which at the time of this writing is 2.4.18. You will need both the USB device filesystem and SCSI generic support. Then, check out the Linux USB Guide (linux-usb.sourceforge.net/USB-guide/book1.html). The “mass storage” section should be helpful.

—Chad Robinson, crobinson@rfgonline.com

New Products

Heather Mead

Issue #99, July 2002

The Instant Wireless Presentation Gateway, GNU X-Tools, Crossbeam X4OS 2.0 and more.

Instant Wireless Presentation Gateway

The Instant Wireless Presentation Gateway from Linksys enables wireless PC users to project presentations and other data onto VGA-equipped devices, such as projectors, monitors and LCD panels, without physically wiring each PC to the projector. Using 802.11b wireless networking, WiFi-enabled users take turns controlling the presentation display by typing in pre-assigned key codes. The Gateway uses an Intersil PRISM 2.5 chipset housed in a MiniPCI slot and runs on a Linux-based OS. Compliant with IEEE's 802.11b industry standard, the Gateway works with compliant wireless devices of any brand.

Contact Linksys, 17401 Armstrong Avenue, Irvine, California 92614, 800-546-5797 (toll-free), sales@linksys.com, www.linksys.com.

GNU X-Tools 2.0

Version 2.0 of GNU X-Tools Standard Edition is available from Microcross. X-Tools 2.0 provides support for over 200 variants from 21 embedded CPU families. Included in X-Tools is a standard tool suite of C/C++, Java and Fortran cross-compilers, the Visual IDE, the Visual debugger, assemblers, binary utilities, a user's guide and 6,500 pages of electronic documentation. Supported target CPUs include ARC, ARM, AVR, H8, i386, i960, M32R, M68K/ColdFire, MIPS, MIPS64, PowerPC, Sparc, Thumb, XScale and more.

Contact Microcross, Inc., 151 Osigian Boulevard, Suite 154, Warner Robins, Georgia 31088, sales@microcross.com, www.microcross.com.

Crossbeam X40S 2.0

Crossbeam Systems' Crossbeam X40S version 2.0 is an open appliance that provides a complete security solution comprised of multiple applications from ISVs that can run concurrently across up to ten application processors. The X40S appliance's binary compatibility provides the ability to run any Linux-based application at line speed on a single device. The X40S is built with redundant interfaces, power supplies and fans, as well as mirrored hard drives, two network processor modules and two control processor modules. Network processor modules can include dual gigabit network interfaces and up to 16 10/100 Ethernet connections.

Contact Crossbeam Systems, Inc., 200 Baker Avenue, Concord, Massachusetts 01742, 978-318-7500, info@crossbeamsys.com, www.crossbeamsys.com.

xSeries Hosting Appliance

IBM's eServer xSeries Hosting Appliance comes with over 40 pre-integrated web development, e-commerce and database applications, allowing rapid installation for telcos and ISPs offering customized service packages. Running Red Hat on Intel-based servers, the Hosting Appliance can host hundreds of sites on a single 1U server. Available in a 1U form factor, it has a Celeron processor with 256MB standard/1.5GB max ECC SDRAM system memory. The 40GB IDE internal storage is standard and can be increased to 80GB.

Contact IBM, 1133 Westchester Avenue, White Plains, New York 10604, 888-746-7426 x7000 (toll-free), www.ibm.com/linux.

Altus 130 Server

The Altus 130 is Penguin Computing's newest addition to its line of 1U server platforms, designed for high-performance cluster computing. The system is built on the QuantiSpeed architecture of the AMD Athlon MP dual processors and the AMD 760MP chipset. It also uses standard PC2100 DDB SDRAM and has two network-bootable 10/100 Ethernet interfaces on the motherboard to provide redundant connectivity in the base system configuration. Other peripheral options are available, including up to two ATA-100 IDE hard drives, various network connections via an open PCI slot and a 24x CD-ROM drive. The systems can be customized as configured for specific needs and can be preloaded with Scyld's Beowulf software.

Contact Penguin Computing, Inc., 965 Mission Street, Suite 730, San Francisco, California 94103, 888-736-4846 (toll-free), info@penguincomputing.com, www.penguincomputing.com.

Linux Carrier-Grade Edition 2.1

MontaVista Software has released MontaVista Linux Carrier-Grade Edition 2.1, a distribution targeted for edge and core telecommunications, including applications for IP and voice networks, optical networks and VoIP gateways. The Carrier-Grade Edition contains high-availability and reliability features, such as kernel and driver hardening, monitoring and fault management services, designed to address specific needs of telecom equipment manufacturers. It also includes PICMG 2.12 hot-swap capabilities, redundant Ethernet and a resource monitoring and event management subsystem.

Contact MontaVista Software, Inc., 1237 East Arques Avenue, Sunnyvale, California 94085, 408-328-9200, www.mvista.com.

Mammoth PostgreSQL

Mammoth PostgreSQL is an SQL-compatible object-relational database management system designed for small to medium-sized businesses. Compatible with PostgreSQL 7.2.1, Mammoth provides a commercially supported distribution for Solaris, Mac OS X and Red Hat x86 platforms. It ships with built-in support for SSL connectivity and programming APIs for C/C++, Perl and Python. Mammoth Deluxe also ships with the SQL/XML application server LXP for integration with web-based programs like Perl and PHP. One-time and subscription-based licensing models are available.

Contact Command Prompt, Inc., 3439 NE Sandy Boulevard, #127, Portland, Oregon 97232, 503-736-4609, info@commandprompt.com, www.commandprompt.com.

Archive Index Issue Table of Contents

Advanced search