Advanced search

Linux Journal Issue #39/July 1997

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

An Introduction to IC Design Under Linux

Toby Schaffer

Alan W. Glaser

Issue #39, July 1997

A discussion of various Linux tools for integrated circuit board design.

As everyone knows, computers are composed of two main elements, hardware and software. There are many well-known, powerful tools for developing software under Linux. What's not so well-known is that there are also tools available for developing hardware. This article introduces some of the freely-available tools you can use to create integrated circuits (ICs). These tools include: Magic, an IC layout editor; SPICE, an analog circuit simulator and Sigview, a graphical signal viewer. When these programs are used in concert, Linux becomes a platform that can be used to create real-world, working chips.

We'll start by by briefly describing the layout and simulation that are the basis of IC design. Next, we'll cover where and how to get the tools. Then we'll go into more detail about what a chip layout represents. We'll also touch on Magic's technology file, and finish by presenting a complete design example to tie everything together.

What Is Magic?

In a nutshell, Magic is a graphical tool that a circuit designer uses to specify how an IC should be constructed. This specification is created by drawing rectangles which represent wires and transistors—the building blocks of most ICs. The rectangles are drawn in various colors and fill patterns which represent the layers used to manufacture the chip. (The “layer” concept is explained in more detail below.) The final drawing is usually referred to as the *layout* of the chip.

Don't be fooled—there's much more to designing an IC than simply making rectangles; otherwise, most any drawing program could do the job. IC design requires further assistance from the software. To that end, Magic is equipped with tools that make IC design easier. For example, it enforces *design rules* which guarantee that the circuit, as drawn, can be manufactured correctly, and it can *extract* a list of circuit components and how they're connected (a *netlist*) from a layout.

These capabilities help provide a basic introduction to Magic and how it's used to do real-world designs. In particular, we'll develop a simple digital circuit—an *inverter*, which is a ubiquitous building block for complex digital chips.

What is SPICE?

Magic is used to draw the rectangles in a layout, but how do we know that the circuit corresponding to these rectangles performs as desired? *Simulation* is a way to verify that the design you've drawn is actually the design you want—it's very much like using a debugger on a piece of software that you're developing. Why bother with simulation? Why not just draw it, build it, and see if it works? Simulation is necessary, because fabricating a chip is costly in a variety of ways. The manufacturing process itself is very expensive, so you want to minimize the number of times you must re-manufacture a defective design. It is also time-consuming, so you want to avoid missing a market window as a result of too many re-manufacturing cycles. Because of these costs, time invested in verifying a design through simulation pays back tremendous dividends.

One tool used to simulate circuit performance is SPICE (Simulation Program with Integrated Circuit Emphasis). Developed at the University of California at Berkeley in the mid-1970's, SPICE and its derivatives, notably PSpice from MicroSim Corporation and HSPICE from Meta-Software, are the most widely-used examples of *analog circuit simulators*. Using detailed models of the individual circuit components, these programs provide an accurate picture of the device's operation and can analyze many different aspects of a circuit's behavior. Later, we'll look at one aspect that is particularly useful in digital circuit design.

Installing Magic

Magic has a WWW page (maintained by Bob Mayo of DEC's Western Research Laboratory) from which you can get source code and browse other relevant information. Point your favorite Web browser to http://www.research.digital.com/wrl/projects/magic/magic.html and follow the “Getting the program and manuals” link. Version 6.5 should be available and out of beta by the time you read this article.

Magic is installed in either the home directory of the user “cad” or the directory pointed to by the environment variable **$CAD_HOME**. You must either create a “cad” user and make its file space world-readable (assuming you want everyone to be able to run the tools), or designate a directory for the tools and set **$CAD_HOME** to point to it. We assume you'll choose the second method and refer to the directory specified by **$CAD_HOME** as the root of the directory tree in which the tools are installed.

After setting **$CAD_HOME**, untar **magic-6.5.tar**. Go to the **magic-6.5** directory and enter **make config**. Choose options **1** (X11), **4** (Linux), **5** (Intel based systems, assuming you're on an Intel). Next is a list of optional modules. Leaving them out trades functionality for smaller executable size and compile time. For now, just accept all the optional modules. You can always recompile later if you find you don't use one feature or another. Then, following the instructions on the screen, do a **make force** to compile everything. After this compile finishes successfully, **make install** installs everything in the correct location. Don't skip this last step, because Magic depends on having all its support files in the right place.

Installing SPICE

The latest freely-available version of SPICE is 3f4. Linux-ready sources for SPICE3f4 are available at ftp://ftp.sunsite.edu/pub/Linux/apps/circuits/ spice3f4.tar.gz. Compilation and installation instructions are in **spice3f4.readme**. Note that you must have at least version 1.14.5 of BASH in order to compile. When we compiled ours, we made only one change. We removed the **-dWANT_X11** from the **conf/linux** file, and so did not get the X interface but did get smaller binaries. Since we are using Sigview to look at the the outputs, we don't need this interface anyway. If you don't feel like compiling, an a.out format binary is available at ftp://ftp.eos.ncsu.edu/pub/vlsi/ software/Linux/spice3/.

Installing Sigview

Sigview, developed by Lisa Pickel at MCNC, is an X11-based program for viewing the output of various simulators, including SPICE. By the time this article appears, it should be available at **ftp://ftp.sunsite.edu/pub/Linux/apps/circuits/ sigview31.tgz**. After untarring this file, **cd sigview** and edit the Makefile. As noted in the instructions at the top, you might need to define **SIG_DEV_FILE** and **SIG_TMP_PATH** to point to appropriate places on your system. If they are already set correctly, you can simply use the included (a.out format) executable. If not, a simple **make** will do the job. You can then place the executable in **$CAD_HOME/bin**.

Layout Concepts

Before getting into the details of designing with Magic, it's very helpful to have a basic understanding of the physical structure of an IC and the process by which it's constructed. There are many different types of process technologies—CMOS, GaAs, bipolar and BiCMOS are common ones. Today, CMOS (Complementary Metal Oxide Semiconductor) is the dominant technology for high-density, low-power, low-cost digital circuits; accordingly, we'll focus our examples on designing for CMOS. However, the concepts presented apply equally well to any process.

The Mask

ICs are constructed by depositing, implanting or growing various materials in patterned layers on a silicon base. Several layers are stacked vertically atop one another; each is separated from the layer immediately above and below by a thin layer of insulating material. It's important to note that a layer can contain many separate objects—for example, it can contain thousands of wires, none of which touch one another.

How do IC manufacturers form the patterns? Typical manufacturing processes are similar to spray-painting, in that they coat the entire IC surface with the material used to make the layer; this obviously precludes the formation of distinct shapes or patterns. If you've ever used a template to spray-paint letters onto a poster or wall, you know that the solution to this problem is to get a piece of shielding, called a *mask*, cut out the areas you want to paint, and spray through the resulting holes. IC process engineers do the same thing on a vastly smaller scale.

Figure 1. Two Layer Chip Layout

Each step in the fabrication process uses a different mask. Suppose we have a chip with two layers of metal wiring. We'll call the layer closest to the silicon base “m1” and the layer closest to the top “m2”. Because m1 and m2 are different layers, m2 wires can cross over m1 wires without making electrical contact. This is like a high road using an overpass to cross a low road. The left side of Figure 1 illustrates, in both an aerial and cross-sectional view, a vertical m2 wire overlapping a horizontal m1 wire. Note that the two are electrically separate. Suppose now that we want to create a point where a signal *can* cross from m1 to m2. To build this structure, we need a mask for m1, a mask for m2, and a mask for *via*s. A via is a hole in the insulating layer, which separates m1 from m2 and allows m2 to flow downward and contact m1, forming an electrical connection. This is shown on the right side of Figure 1; the black square is the via, which connects the two layers. Transistors are more complicated, requiring more masks and different materials.

Abstract Layers

Magic makes work easier for the designer by hiding much of this underlying physical complexity. Rather than work with mask layers directly, you manipulate abstract layers which implicitly represent one or more masks. For example, to create an m1 to m2 via, rather than draw three shapes on separate mask layers (the m1 layer, the hole layer and the m2 layer), you simply draw one shape on the abstract “via” layer, which has all three mask layers “built-in”. There are also abstract layers having a one-to-one correspondence to mask layers; m1 and m2 are examples. In general, an abstract layer can represent an arbitrary number of mask layers, permitting a great simplification in design.

Lambda Rules and Scalability

In order to cram as much as possible onto the IC, modern processes define design rule measurements in microns, mm. A micron is one millionth of a meter. For example, in one particular CMOS process m1 wires must be at least 0.8mm apart from each other and at least 0.6mm wide. Each rule is independent of the others and is specified in fractions of microns.

Magic design rules, on the other hand, are expressed not in microns but in multiples of a unit called *lambda*, l. Introduced in Carver Mead and Lynn Conway's classic book *Introduction to VLSI Systems*, lambda is the “quantum” for the rules—all spacings and widths must be multiples of lambda. The previous rules concerning m1 spacing and width would be expressed in terms of lambda as 3<<tt>l and 2[lambda] respectively, where <<tt>l = 0.3<<tt>mm.

Note that using lambda rules is a little costly in terms of area. The spacing rule is now effectively 0.9~<<tt>mm instead of 0.8~<<tt>mm; however, basing rules on lambda allows for *scalable* designs. The scalable rules allow us to create one design and be confident that it is valid (i.e., it violates no design rules) regardless of the value of lambda. For example, the MOSIS service, which is a low-cost, low-volume IC prototyping service, fabricates chips through several different vendors in various processes. These processes have values of lambda ranging from 0.3<<tt>mm to 1.0<<tt>mm; MOSIS basically has one set of design rules which covers all of these processes. In reality, once lambda starts getting below 0.5~<<tt>mm, even scalable rules change somewhat, but this is a detail. These rules are referred to as the SCMOS (Scalable CMOS) rules, and we use them in our design example.

What the Numbers Mean

The Technology File

As mentioned earlier, there are several different IC process technologies. Magic is technology-independent and can be used with all. The information that Magic needs about the target process resides in an external file called the *technology file* (tech file), which Magic reads when the program starts. By default Magic looks in **$CAD_HOME/lib/magic/sys** for the file scmos.tech27. Both the location and name of the tech file can be specified on the command line with the **-T** switch, although the name of the tech file must end with a ".tech27" extension. For example, to use the scmos-sub.tech27 tech file, you type **magic -T scmos-sub**.

Included with the Magic distribution in the **scmos** directory are multiple tech files for various processes offered by MOSIS. As mentioned above, when lambda gets very small, the design rules begin to change. MOSIS provides different tech files to account for this.

Items defined in the tech file include:

- layer definitions (e.g., names, colors) and how layers interact with one another
- design rules
- device (transistor) geometry and parasitics
- instructions on how to convert Magic's abstract layers to mask layers and vice versa

Knowing the tech file internals isn't necessary to use Magic; in fact, unless you're writing a tech file for a new process or modifying one for an existing process, you can think of the tech file as a "black box". If you're interested, you can find a complete discussion of the tech file format in *Magic Maintainer's Manual #2: The Technology File*, located in the **doc** subdirectory of the Magic source tree.

Design Example

We now pull it all together with a real-world design example—an inverter. An inverter simply flips a bit; it turns a **1** into a **0** and a **0** into a **1**. This may sound trivial, but it's an essential building block for more complex digital circuits.

First, we'll walk through the layout and design rule check (DRC) of the inverter, explaining things as we go. Then we'll extract the netlist and simulate the chip using SPICE. After we're convinced the chip works properly, we'll construct the CIF file to send to the foundry to have the chip fabricated.

Although this example is very rudimentary, you can find much more complete documentation in the *Magic Tutorial* series, located in the **doc** subdirectory of the Magic source distribution. These excellent tutorials cover both the basics and more advanced usage (such as hierarchical design, multiple windows and interactive routing) that isn't covered here.

Introductory Magic Commands

First, invoke Magic from your shell prompt. Let's call our example file "inverter". We'll use the default tech file, so we don't need to specify one with the **-T** switch.

```
magic inverter
```

A new *layout window* appears in which you design your chip; error messages and feedback appear in the window in which you started Magic (the *command window*). Magic handles window focus a little differently than you might expect—even though the input focus remains on the layout window, anything you type appears in the command window. Many commands work regardless of which window has the focus. When everything is ready, Magic displays a prompt in the command window.

The first thing to do is "paint" the two metal wires that supply current to the inverter—these wires are called the *power rails*. To create the first rail, make sure the focus is in the layout window, and type the following commands (the colon in the first column is required):

```
:grid
:box 0 0 4 5
:paint m1
```

The first command simply turns on the reference grid, although we're zoomed too far out to see it right now. The second command changes the shape of the *box* so that the lower-left corner is at (**0,0**) and the upper-right corner is at (**4,5**). The third command fills the box with blue paint that represents the first layer of metal. The box is the center of attention; most operations, such as painting and erasing, affect only the area within the box.

Now zoom in to get a clearer picture:

```
:box 0 0 12 32
```

Instead of typing a long command, we use a macro. Simply press the **z** *without* the colon, and the layout window zooms to the box. Note that after zooming the reference grid appears; each hash mark represents one lambda. Shortcuts like this are very handy, so let's explore them some more.

Shortcuts—Macros and the Mouse

A macro is a single keystroke not preceded by a colon. It can represent any arbitrary sequence of typed commands; for example, the **z** above represents the **:findbox zoom** command. System-wide macros are read at start-up from the file $CAD_HOME/lib/magic/sys/.magic. You can define you own macros in a **.magic** file in your home directories. And you can define macros at any time by entering:

```
   :macro <key> <action to perform>
```

The mouse provides a quick way to both move and resize the box as well as paint. Move the cursor around the layout window and click the left mouse button as you do. This action moves the box so that its lower-left corner (the anchor point) is placed at the spot where you clicked. Now move the cursor around while clicking the right mouse button. The corner opposite the anchor point is placed where you let the button up. Notice this might result in the anchor point moving, since it's always the lower-left corner of the box.

To see how the mouse makes painting easier, first enter:

```
   :box 0 27 4 32
```

to move the box where we want the second rail. Normally, you'd use the mouse to move and resize instead of typing a **box** command, but we want to make sure we all have the box in the exact same location. Now instead of typing **:paint m1**, move the cursor over the existing m1 rectangle and click the middle mouse button (for those of you using a 2-button mouse, click both buttons simultaneously). The box is now filled with metal 1. Clicking the middle button fills the box with whatever paint layers are underneath the cursor when you click. This leads to a simple way to erase any paint in the box—place the cursor over empty background and click the middle button, which paints “nothing” into the box.

Arggh—I Made a Mistake

Like any good editor, Magic has an undo facility for those times when you make a mistake or just want to try something out without having to commit. The **u** macro undoes the last action while the **U** macro implements the redo command. It's not unlimited; expect only the last 10 actions or so to be undoable. However, it's more than enough to let you erase a few rectangles and get them back.

Drawing the Inverter

A CMOS inverter has two Field-Effect Transistors, one p-type (PFET) and one n-type (NFET). Transistors are constructed by drawing *polysilicon* (or poly, for short) over *diffusion*. There are two types of diffusion, p and n. A PFET is poly over p-diffusion; an NFET is poly over n-diffusion.

Now let's start drawing the transistors. Position the box at ll (lower left) = (**4,19**), ur (upper right) = (**8,27**) and type:

```
:pai pdiff
```

Magic understands abbreviations for commands, such as **pai** for **paint**. The layer **pdiff** is p-type diffusion; each layer can have multiple names defined in the tech file. In SCMOS, p-diffusion can be called **pdiffusion**, **pdiff** or **brown**.

Now it's time for the n-diffusion. Place the box at ll=(**4,5**) and ur=(**8,9**) and type:

```
:pai ndiff
```

At this point your layout should look like the one in Figure 2. If not, you've gone astray; use the undo command and try again.

Figure 2. Layout Showing Diffusions Only

The transistors need poly running over the diffusion. In an inverter, the two polys for the transistors are connected together. This is the inverter's input—the signal driving the inverter is connected to the poly. It's simplest, then, to draw a single piece of poly which crosses both diffusions. Place the box at ll=(5,3) and ur=(7,29) and type:

```
:pai poly
```

Your layout should then look like Figure 3.

Figure 3. Layout Showing Polysilicon Crossing Diffusions to Define Transistors

For FETs, Size Does Matter

There are a couple of important points to notice. First, the area where the poly crosses the diffusion has a diagonally-striped pattern different from both poly and diffusion. This area represents the actual transistor, since FETs are constructed wherever poly passes over diffusion. To see this, place the cursor over the top transistor, and select it by pressing the **s** key. Now enter **:what**, which displays the names of the selected paint. Here, the selected area is “ptransistor”; Magic actually changes the paint from pdiffusion to ptransistor when poly crosses over it. The same concept applies when poly crosses over n-diffusion to form an “ntransistor”.

The other, very important thing to notice in Figure 3 is that white dots suddenly appear. If you work with Magic, get used to these dots—they indicate design rule violations, which Magic always checks for whenever something changes. To see which rule has been violated, place the box over the error dots, and use the **y** macro, which represents the **:drc why** command. Magic prints the reason for the error in the command window:

```
Diffusion must overhang transistor by at least 3 (MOSIS rule #3.4)
```

Since the overhang is only 1, we get design rule violations. Notice the dots occur only in the area that causes the error.

Now that we know what the problems are, let's fix them. Move the box to ll=(**2,19**) and ur=(**10,27**) to contain the upper error dots and type **:pai pdiff**. The error disappears. Now color the area in ndiffusion to eliminate the remaining errors. The layout now contains two “DRC-clean” transistors—no error dots are now displayed.

Connectivity

To see which components are connected to a particular rectangle, place the cursor over it and press the **s** key twice quickly. The first key press selects the rectangle; the second selects everything that's electrically connected to it. Try this on either metal rail, and notice that they are both electrically isolated from everything else. The top rail must connect to the diffusion on one side of the PFET and the bottom rail to the diffusion on one side of the NFET; otherwise, no current can flow. Thus, we need what's called a diffusion-to-m1 contact, abbreviated *pdc* or *ndc* depending on whether we're talking about p or n diffusion respectively.

Place the box at ll=(**0,5**), ur=(**4,9**) and type **:pai ndc**. Now make a pdc at ll=(**0,19**) and ur=(**4,27**). Checking connectivity shows that the rails are connected to both the adjacent contacts and the diffusion right up to, but not including, the transistor. At this point, check your layout against Figure 4.

<u>Figure 4. Layout with Electrical Connections Added</u>

Now for the output. In a CMOS inverter, the output is made by connecting together the sides of the transistors that are not connected to a rail. Place the box at ll=(**8,5**) and ur=(**12,9**) and paint an ndc (try doing this using the middle mouse button). Then put the box at ll=(**8,19**) and ur=(**12,27**) and paint a pdc. To connect these two contacts together, we need m1 between them: put the box at ll=(**8,9**) and ur=(**12,19**) and paint m1. Check the connectivity of the output to verify that the two contacts are tied together.

Congratulations. You've just drawn the layout for a fully-functional CMOS logic gate—however, we're not quite finished.

Adding Labels

Labeling assigns a name to a specified rectangle in the layout and thereby to every rectangle electrically connected to it. Labeling serves two purposes. First, it's like commenting code in that it lets you know what signals are where. Second, it makes simulation much easier when you pick sensible names rather than generate them automatically; for automatically, “input” is much easier to remember than “a_43_n15#”.

Let's label the rails first. Select the upper rail by moving the cursor over it and pressing the **s** key. Then type:

```
   :label Vdd
```

“Vdd” should appear next to the rail. Select the other rail and type **:lab Gnd**. Notice the labels are placed somewhat randomly. Now select the poly between the two transistors and type **:lab in center**. This places the label in the middle of the selected rectangle. Finally, select the output m1 and type **:lab out center**. The now-finished layout should look like Figure 5.

Figure 5. Completed Layout with Labels

To save your layout, type **:save**. If you want to save it under another name, simply type the name too; e.g., **:save** ***newcellname***. Magic automatically appends a **.mag** extension.

Netlist Extraction

The next step is to simulate the layout to verify that it works correctly. Magic's **.mag** files are stored simply as a collection of various types of rectangles, but simulators require a netlist of circuit components, such as transistors and capacitors. Recall that a netlist is a list of circuit components and how they're connected.

You use Magic itself to extract a netlist from a layout. Magic's extractor recognizes various combinations of rectangles as defined in the tech file and converts them into the appropriate circuit elements. It also extracts connectivity between shapes, thus making a complete netlist.

You must choose an “extraction style”, which tells Magic how to interpret the shapes in the layout. To see the available styles, type **:extract style**. For our purposes, the important part of the style name is the number, which refers to the minimum transistor length. For example, in the **lambda=1.0(scna20_orb)** style we'll use, this length is 2.0**m**m. This should be the current style; if it's not, enter:

```
:extract style lambda=1.0(scna20_orb)
```

To complete the extraction, simply type **:extract**. Magic makes an **inverter.ext** file. This file is an intermediate description of the circuit containing all the information necessary to build a netlist for various simulators. We're now ready to begin simulation, so quit magic by entering **:quit**.

Simulation

Before beginning a simulation, first create a SPICE file and add transistor models and SPICE commands to it. Next, simulate the inverter to verify that it was laid out correctly.

Creating the SPICE File

First, translate the **inverter.ext** file into something that SPICE can understand. The ext2spice program by Stefanos Sidiropolous that comes with the Magic distribution performs this translation. Simply enter:

```
ext2spice -f spice3 inverter.ext
```

We specify “spice3” format output, because earlier SPICE versions can't handle text strings for labels.

Transistor Models

We now have a file that SPICE can understand, but it's incomplete in an important way. For SPICE to simulate a device correctly, it needs a *model*, a mathematical description of the device's behavior. In particular, we now need models for the two transistors (n-type and p-type) that we've used in our design.

SPICE has a variety of built-in transistor models specified in terms of sets of parameters. These parameters vary according to the fabrication process used, so it's up to the user to specify the correct parameters for the process being used. Fortunately, you don't have to figure these out yourselves—you can get them from MOSIS at ftp://ftp.mosis.edu/pub/mosis/vendors/orbit-scna20. This FTP site contains a whole slew of data from past runs; we chose a typical one from the “Level 2 Parameters” section. One minor detail—note that you need to change the “CMOSN” and “CMOSP” to match ext2spice's output, “NFET” and “PFET”.

After pasting in the transistor models, the inverter.spice file should look like Listing 1.

Digital vs. Analog

We've drawn the inverter and extracted a netlist. Now we need to simulate the chip to make sure it'll perform as expected when the chip is fabricated. A simulator such as SPICE can do many different types of circuit analysis. We will demonstrate two that are very useful for the digital circuit designer.

It might seem curious to use an *analog* circuit simulator for a *digital* circuit. We think of digital circuits as being in one of two states, **0** or **1**, while analog circuits can take any of a continuous range of values. What's important to realize is that a digital circuit is actually a special kind of analog circuit. At the circuit level, a signal is either a voltage or current, and these are really analog quantities. The conventional **0** and **1** representations of a bit are simply abstractions of two particular voltage values; the actual values depend on the process in which the chip is fabricated. With CMOS in particular, you can generally ignore the currents and deal only with the voltages, due to the FET's mode of operation. For example, in the particular process for which we extracted our inverter, a **1** corresponds to 5 Volts (5V). In other processes, it might correspond to 3.3V or 2.5V. Happily enough, a **0** generally means 0V. As we all know, digital circuits don't switch instantaneously between their two values. It takes time to change voltages, and this delay is one of the things that makes your CPU run at 100MHz instead of 133MHz.

The upshot is that SPICE simulates analog behavior, but as digital circuit designers, we will interpret the circuit's output as digital bits rather than the analog voltages they actually are.

Simulators specialized for digital circuits do not provide the level of detail, that is, accuracy, that SPICE does, but they are orders of magnitude faster. IRSIM, which is available through the Magic WWW home page (see Resources at the end of the article) is an example of such a simulator.

Transient Analysis

Suppose we apply the test vector **101** to the inverter over a period of 75 nanoseconds (1 ns = 10-9 seconds). How will the output change over this time interval? To answer this question, SPICE can do a *transient analysis*. In a transient analysis, you define the input signal(s) and tell SPICE to simulate the circuit for a certain amount of time from the circuit's point-of-view, not real time. A transient analysis provides timing information, for example, how long the output takes to switch from **1** to **0**. This analysis is also useful for verifying the circuit's functionality; you simply apply the test vector and check the output for correctness.

Let's apply the vector **101** to our inverter. Add the following lines to the end of the **inverter.spice** file:

```
Vpwr Vdd Gnd 5
Vgnd Gnd 0 0
Vinput in Gnd 0 PULSE( 0V 5V 0ns 2.5ns 2.5ns 25ns 50ns )
 .TRAN 0.1ns 75ns
 .end
```

Let's go over this specification line-by-line. Line 1 defines a 5V voltage source between **Vdd** and **Gnd**; this is the power supply. Line 2 is necessary because of a SPICE artifact of always referring to ground as **0**. We simply tie **Gnd** and **0** together (electrically speaking) by using a 0V voltage source, i.e., a short-circuit. Line 3 defines a voltage source between “in” and ground; this is the inverter's input. Looking at the parameters in parentheses, we have a voltage source whose initial voltage is 0V and “pulsed” voltage is 5V, and it starts after a 0ns delay. It has a *rise time* (i.e., the time required to switch from **0** to **1**) of 2.5ns, and a *fall time* (i.e., the time required to switch from **1** to **0**) also of 2.5ns. The width of each pulse is 25ns, and it repeats after 50ns have elapsed. Line 4 means that we want to do a transient analysis with a resolution of 0.1ns that lasts for 75ns. Line 5 ends the circuit file. Save the file, then run the simulation by typing:

```
spice3 -r inverter.out < inverter.spice
```

Now, let's view the results by typing:

```
sigview inverter.out
```

We're interested only in the top two signals, “out” and “in”, as shown in Figure 6.

Figure 6. SPICE Inverter Output

From the digital point-of-view we see that the inverter correctly produces a **010** pattern; from the analog point-of-view we see that the output's rise time is about 0.91ns and its fall time is about 0.89ns.

Making the Masks

At this point, the layout is completed and verified, and it's time to send it off to the foundry to be manufactured. Since Magic files don't contain any information about physical dimensions (remember, all measurements are in terms of lambda), we need to create a file that gives the layout's shapes definite sizes in terms of microns. Also, since this file is used by the foundry to pattern the masks used to make the chip, it specifies shapes in terms of mask layers instead of Magic's abstract layers. Magic understands two file formats for describing physical geometries, CIF (Caltech Intermediate Format) and Calma GDS-II; MOSIS accepts both. We arbitrarily chose CIF for our example.

Just as there are several extraction styles, there are several CIF styles. The first thing we need to do is specify the correct one. Launch Magic again with the inverter file (type **magic inverter** at the shell prompt), and then type **:cif ostyle** to see a list of available CIF output styles. The current style should be **lambda=1.0(nwell)**; if it's not, make it so by typing **:cif ostyle lambda=1.0(nwell)**.

Creating the CIF file is simple; type **:cif write inverter**. This creates the file inverter.cif, which we'd send to MOSIS. This process is referred to as “tapeout”, a term coined before the advent of FTP when IC designs were stored on magnetic tape. If this were a real design, you would now take to your bed to make up for the fact that you hadn't slept in the last three weeks.

Conclusion

We've introduced three powerful tools for IC design under Linux:

1. Magic, for creating layouts
2. SPICE, for simulating circuits extracted from the layouts
3. Sigview, for viewing the results of SPICE simulations.

With these tools, a designer can create working, commercial-quality chips without spending lots of money on a workstation and CAD software.

The design example we used to demonstrate these tools was small but not useless. In fact, Figure 7 shows a 32,701-transistor IC measuring 2.71mm by 6.15mm, designed with Magic, that uses building blocks very much like the inverter we just made. (This may sound like a lot of transistors, until you consider that current commercial microprocessors are rapidly approaching 10 *million* transistors on a chip smaller than 2cm by 2cm.)

Figure 7. Magic-Designed IC

Thanks for making it this far. Obviously, there's a lot we've left out about the complexities of hardware design. However, we have demonstrated that Linux can be used for developing hardware as well as software. Perhaps the “SuperGizmo 6000” will be designed on the Linux boxes of the future.

Further Reading

We've barely scratched the surface of IC design. If you're interested in exploring this area further, you'll want to consult some references. We have used and can recommend the books listed in the Resources box.

Resources

Toby Schaffer is an electrical engineering student in the Ph.D. program at North Carolina State University. To impress women, he tells them his research is on clocking high-speed multi-chip module digital systems, which might explain why he doesn't have many dates. Article comments and questions are welcome at jtschaff@eos.ncsu.edu.

Alan W. Glaser is working toward his Ph.D. in electrical engineering at N. C. State University. He's interested in physical design of ICs and systems and is currently trying to nail down a dissertation topic so he can graduate. In his spare time, he likes to hang out with his wife and two cats. He can be reached at awglaser@eos.ncsu.edu.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Analyzing Circuits with SPICE on Linux

Kevin Cosgrove, P.E.

Issue #39, July 1997

All about SPICE: how to get it and what to do with it.

SPICE is the Simulation Program with Integrated Circuit Emphasis, first released from the University of California at Berkeley in the early 1970s. Before the existence of SPICE engineers designed circuits by hand, possibly with the aid of a slide-rule or calculator. A prototype was constructed from the original design, and its performance evaluated against the designer's goals.

Designing many of today's circuits would be impossible without the aid of SPICE. Frequently analog circuits contain hundreds or thousands of devices. Design and analysis involve finding solutions to simultaneous equations. These equations can be of simple algebraic form or involve nonlinear differential equations. Prototypes are still constructed to gauge performance, but given costs running in the hundreds of thousands of dollars, performance must be largely anticipated through computer simulation before prototype fabrication begins.

SPICE is not limited to integrated circuit design. Rather, SPICE is useful for analyzing any circuit which can be described in terms of voltage sources, current sources, resistors, capacitors, inductors, transistors and a few other components.

Where to Get SPICE for Linux

SPICE version 3f4 was released in 1993, and the source code is freely available to those friendly to the U.S.A. You can get a copy of the source code from ftp://sunsite.unc.edu/pub/Linux/apps/circuits/spice3f4.tar.gz.

I run Red Hat Linux and use the Red Hat Package Manager (rpm). If you're running rpm, you can take advantage of Andrew Veliath's spice-3f4-2.src.rpm package. You can find a copy on ftp.redhat.com and mirrors. If you don't use rpm, you might want to consider building it and using rpm2cpio to unpack the spice-3f4-2.src.rpm package, since it contains two very useful patches to the pristine source for building on Linux systems.

How to Build/Install SPICE on a Linux System

If you're using rpm, building SPICE is as easy as:

```
rpm -ba -vv SRPMS/spice-3f4-2.src.rpm
rpm -i -U -vv RPMS/i386/spice-3f4-2.i386.rpm
```

The first line builds the installable package from the source package. The second line installs the package and updates the rpm database.

If you're not using rpm, building and installing SPICE is a little more involved but not too bad. The basic process is as follows:

```
rpm2cpio SRPMS/spice-3f4-2.src.rpm | cpio -i
tar xzpf spice3f4.tar.gz
patch < spice3f4.newlnx.patch
patch < spice3f4.dirs.patch
cd spice3f4
util/build linux
```

Compiling took 12 minutes on my 200MHz Pentium system.

```
util/build linux install
strip /usr/bin/{spice3,help,nutmeg,sconvert,multidec,\
        proc2mod}
install -m 644 man/man1/spice.1 /usr/man/man1
install -m 644 man/man1/nutmeg.1 /usr/man/man1
install -m 644 man/man1/sconvert.1 /usr/man/man1
install -m 644 man/man3/mfb.3 /usr/man/man3
install -m 644 man/man5/mfbcap.5 /usr/man/man5
```

After installation comes the real fun—creating and simulating circuits. While a minimal Linux system can run SPICE adequately, analysis time can be significantly improved when it is run on a system with a fast processor(s), 133MHz or above. Additional RAM is equally important, especially for larger circuits. I have 32MB in my home Linux/Pentium system and 128MB in my SunOS/Sparc20 at work.

Constructing a Simple Circuit

Figure 1. Differential Pair Circuit Schematic

Figure 1 shows a the schematic of a differential pair circuit constructed from bipolar junction transistors and resistors. This circuit can be used for either digital or analog purposes and, in either case, can be simulated using SPICE. The circuit operates in the following manner. Very little current flows through the base of transistor Q2, so the base can be considered to be held near ground potential, zero volts. When the input voltage, VIN, is low, near ground, Q1 will be off and Q2 will be on. No current will flow through Q1, so VO1 will be high, equal to VCC. All of the current in IEE will flow through Q2. The voltage drop across RL2 will be:

$5k\Omega*1mA=5\_V$

So the voltage VO2 will be 5V below VCC. So, VO2 will be 0V.

In linear analog operation VIN will be held near ground except for small signal excursions away from ground. Under this condition the differential pair will serve as an amplifier where the voltage gains are:

```
AV1 = VO1 / VIN = -(gm1/2) * RL1
```

```
AV2 = VO2 / VIN = (gm2/2) * RL2
```

where gm1 and gm2 are the transconductances of the two transistors. These values can easily be calculated by hand, but since the point of this story is to show SPICE at work, we'll let SPICE tell us the transconductance values.

Listing 1

Listing 1 shows the SPICE input file corresponding to the circuit in Figure 1. The SPICE input file contains a description of the circuit and its connections, input stimuli, statements to control what kind of analysis SPICE will perform, statements to control output, comments and a title. The first line is always the title and the last non-blank line is always **.end**. Comment lines begin with an asterisk (*****). Control lines of any kind begin with a period (**.**). The line continuation character is a plus sign (**+**) which goes at the beginning of a line being continued from the previous line. This is a little different from the common backslash (\) line continuation, used elsewhere in Linux, where that continuation character goes at the end of the line being continued on the next line.

Line by Line

The options lines shown here:

```
    .opt nopage
    .width in=72
    .width out=80
```

specify that no page breaks will be in the generated output text and the line length of the input and output text. The first line below instructs SPICE to perform a DC analysis of the circuit where VIN is run from -0.15V to 0.15V in 0.010V increments. The next line tells SPICE to generate operating point information for the circuit. It is this line that reports the transconductances of each transistor.

```
  .dc vin -0.15 0.15 0.010
  .op
```

The middle half of the input file describes the connection of circuit elements and sources. In SPICE connection points are termed *nodes*. Every SPICE circuit must contain a ground node numbered zero.

```
  iee 3 vee 1m
```

iee is a current source, known to SPICE by the leading **i**, where **1mA** of current flows from node 3 toward the vee node.

```
  vin input 0 0 sin(0 0.3 5meg) ac 1
```

vin is the input voltage source connected at the positive end to the input with the negative end connected to ground. The DC value of the input voltage is 0V. The time varying—“transient” in SPICE jargon—portion of the input voltage is a sine wave centered at 0V with an amplitude of 0.3V oscillating at a frequency of 5MHz. The AC portion of the input will be normalized to 1V. That is, the AC analysis in SPICE doesn't exercise large signal behavior of the circuit. Digital behavior is the extreme of large signal behavior. Large signal performance of a circuit can be simulated using transient analysis in SPICE.

```
  rl1 out1 vcc 5k
```

rl1 is a resistor connected between the out1 and vcc nodes with a value of 5k.

```
  q1 out1 1 3 bjt
```

q1 is a **bjt** model instance with collector, base and emitter connections at nodes out1, 1 and 3 respectively. The model definition below names a model **bjt** of the type **npn** and with specific parameters. SPICE knows about **npn** transistors and a number of other types of circuit elements. This definition makes use of line continuation.

```
  .model bjt npn(bf=80 rb=100 ccs=2pf
  +   tf=0.3ns tr=6ns cje=3pf cjc=2pf
  +   va=50)
```

The **.plot** line in Listing 1 tells SPICE to plot the voltage values at nodes out1 and out2 calculated during the DC analysis.

Analysis of the Circuit and Interpreting the Results

This command will run a SPICE analysis using the input file from Listing 1 named diffpair-1.cir:

```
spice3 -b diffpair-1.cir
```

Listing 2

The **-b** option causes SPICE to run in batch mode. Listing 2 shows the output of the SPICE analysis. The operating point information gives the DC bias voltages for all the nodes in the circuit and the current through every voltage source. A customary trick to measure current is to insert a voltage source whose voltage is zero. This does not hinder simulated performance, but the circuit will simulate slightly slower, since there's more in it. Model parameters are reported for each type of circuit model used in the simulation. Operating characteristics for the two **bjt** instances show the transconductance of each transistor to be 0.0191 A/V.

Figure 2. Output Voltages

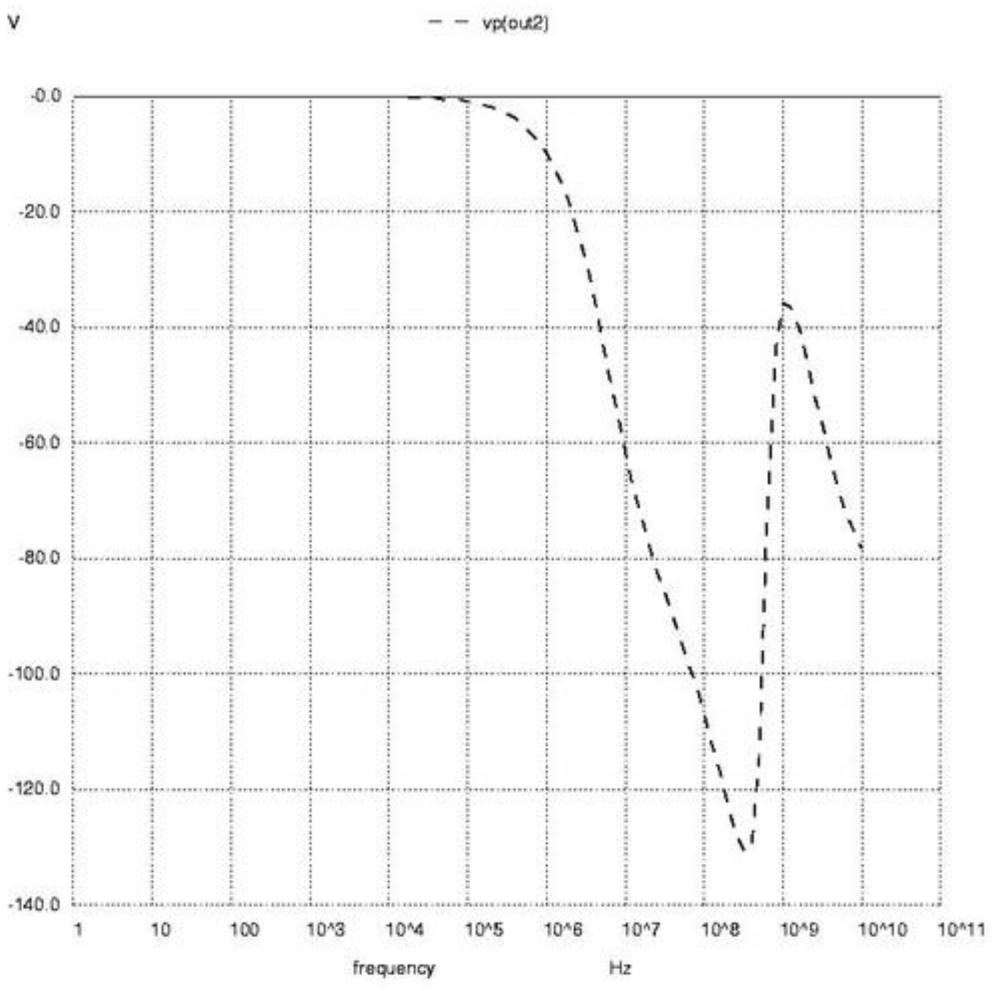


Figure 3: AC Analysis Plot in Degrees

The DC transfer characteristic plot shows output voltage, from left to right, ranging over 0V to 5V and shows the input voltage, from top to bottom, ranging from -0.15V to +0.15V. Each time I view a plot like this I get nostalgic, recalling the days when I would flip switches on the front panel of a PDP-8 and load boot programs from paper tape. But that's another story. This plot certainly isn't fancy by today's standards, but it does convey the necessary information.

Finally, SPICE reports that the simulation took just under 1/10th of a second to run.

Listing 3

Listing 3 adds more analysis directives to the SPICE input file from Listing 1. The analyses to be performed are DC, transfer function, AC and transient. This time we'll start SPICE with an option to save the resulting data to a file, like so:

```
spice3 -b -r diffpair-2.raw diffpair-2.cir
```

After running SPICE the data file can be perused with **nutmeg**, which comes with the SPICE package. Start nutmeg simply by running

```
nutmeg diffpair-2.raw
```

Listing 4

Listing 4 captures an interactive session with nutmeg. At the first nutmeg prompt I entered **setplot** to give me the names of analysis data sets within the raw data file. I selected **dc1** and then plotted the output voltages, shown in Figure 2. When plotted in this way, you can zoom in on sections of the plot using the right mouse button to define the zoomed plot's borders. You can also find the coordinates of any point on the plot or pair of points by either clicking or clicking and dragging the left mouse button. The coordinates display in the window where nutmeg runs. You can put labels on the plot by typing with the keyboard. By setting the type of hard copy device appropriately you can save the plot in a file for later printing, as shown in nutmeg commands 3 through 5.

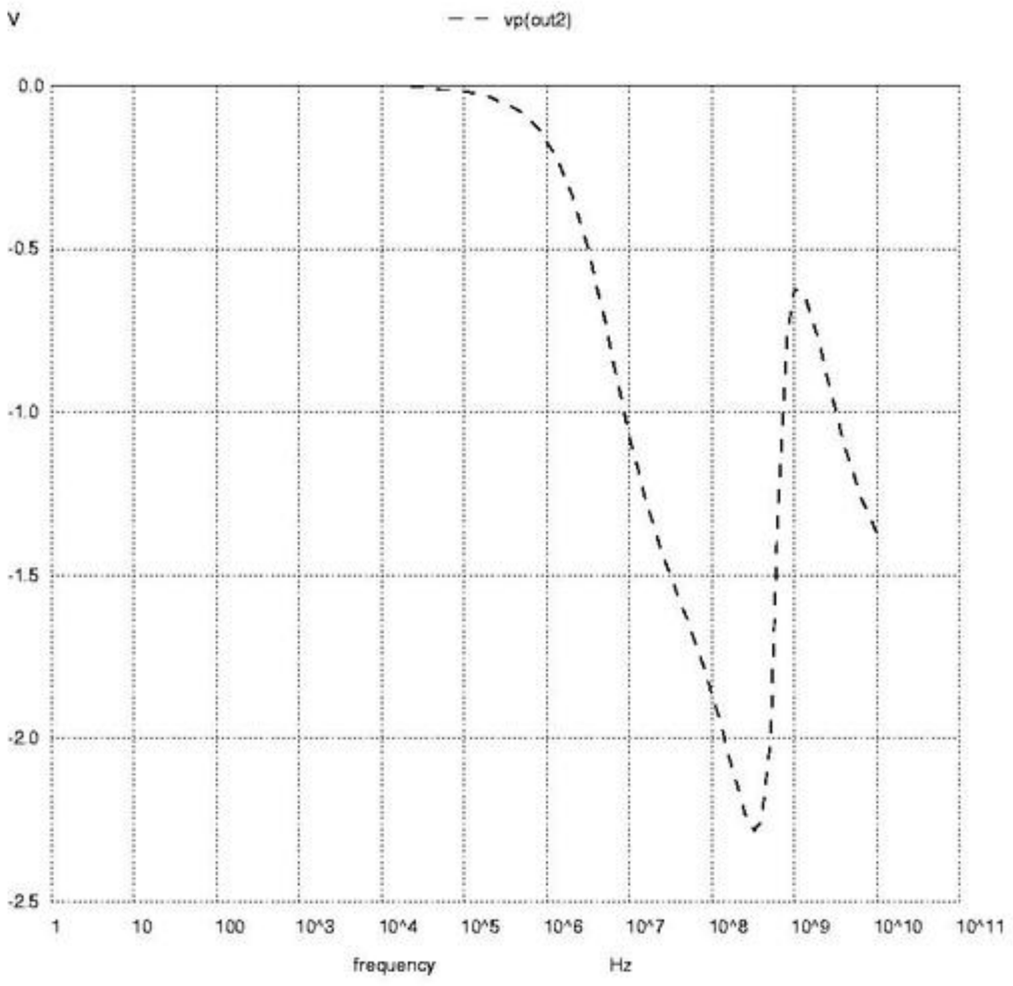


Figure 4: AC Analysis Plot in Radians

Next, in Listing 4, I selected the transfer function data. The display command shows me the variables I can query from the transfer function data. At prompt number 9 I asked to see the transfer function that SPICE reports to be 44.60971. This differs somewhat from the idealized gain equation, which neglects output resistance of the **bjt**.

$$A_{V2} = (g_{m2}/2) * R_{L2} = (0.0191/2)(A/V) * 5k\Omega = 47.75$$

Rather than just selecting one variable for display, at prompt number 10 I selected all variables for display. This shows the output impedance to be about.

$4.9k\Omega$,

, and the input impedance to be about

$8.9k\Omega$.

Nutmeg prompts 11 through 16 create the AC analysis plots of voltage phase at node out2 in degrees, voltage phase at node out2 in radians and magnitude of the voltage at node out2. These are shown in Figures 3 through 5, respectively.

Prompts 17 through 19 resulted in Figure 6, which shows the input voltage sine wave and the output voltage sine wave. Notice that the output voltage shows distortion of an over-driven amplifier.

At prompt 20 I ran a Fourier analysis on the time domain voltage to find the harmonic content of the distorted sine wave. I had to specify the fundamental frequency as 5MHz, the same as was given in the input file, and the node voltage the Fourier analysis should inspect. As expected from the compressed shape of the output sine wave, the total harmonic distortion (THD) is quite high.

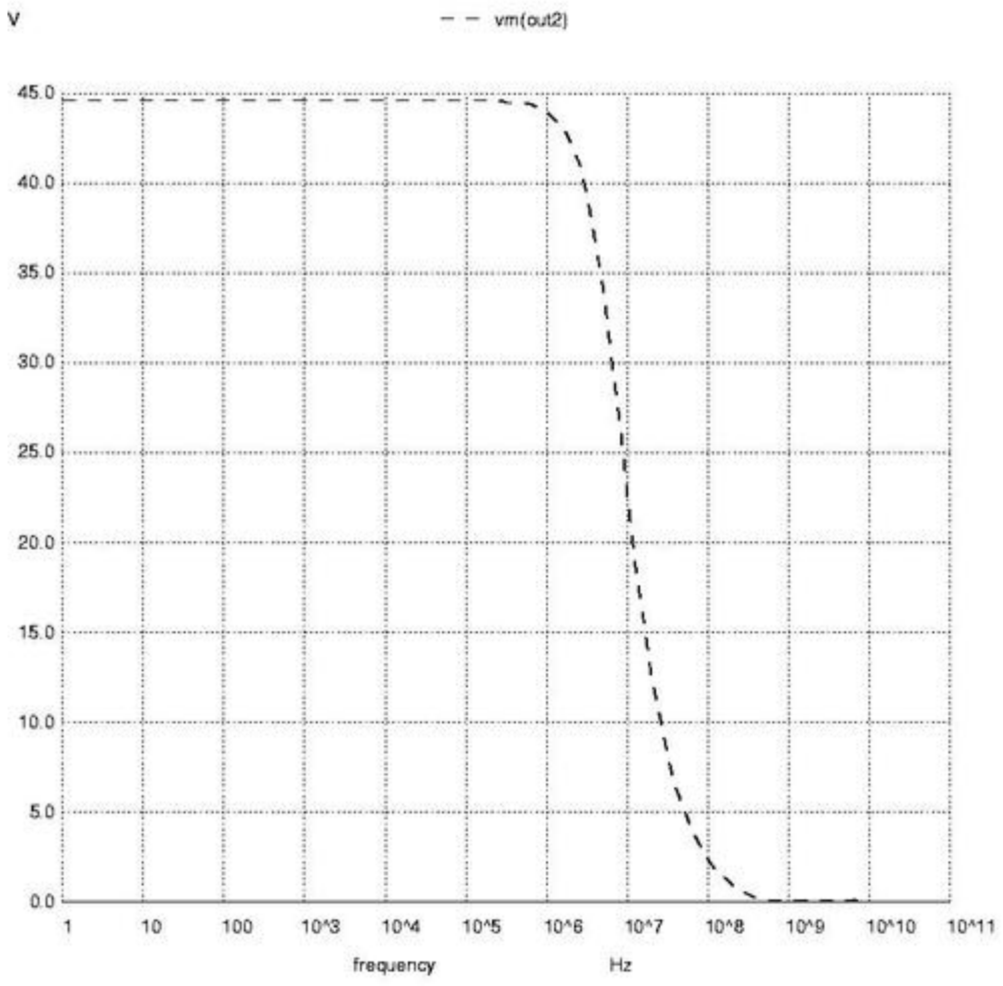


Figure 5: AC Analysis Plot as a Function of Magnitude

Convergence and Accuracy

If you're going to have trouble with SPICE, most likely it will be with a circuit you can't analyze. The good news is that SPICE3 is improved in this respect, and you have some control over how numerical solutions are determined. When SPICE calculates node voltages and branch currents, it uses thresholds for tolerable errors to determine when a simulation reaches its answer—that is, when it reaches numerical convergence. The three parameters controlling the thresholds can be set on the **.options** statement and are named ABSTOL, VNTOL and RELTOL. ABSTOL is the smallest current you want SPICE to accept. Increasing ABSTOL from its default value of 12pA can help a simulation to converge. VNTOL is the smallest voltage that you want SPICE to accept. Increasing VNTOL from its default value of 10V can help a simulation to converge. RELTOL is the ratio of the numerical answer found during the present iteration to the numerical answer found during the last iteration. Increasing RELTOL can help a DC analysis to converge, but increasing RELTOL can also cause transient analysis problems. If you get a warning from SPICE saying "timestep too small", RELTOL is probably set too large.

The parameters ITL1 through ITL6 control the number of iterations to perform before SPICE gives up, and control methods are used to attain convergence.

Obviously the accuracy of the simulation results can be no better than the convergence thresholds used during analysis. If you don't need to relax the thresholds, this won't present a problem since the tolerances on component values and variations in component performance stand to present much more discrepancy between nominal simulated performance and real-world measured performance.

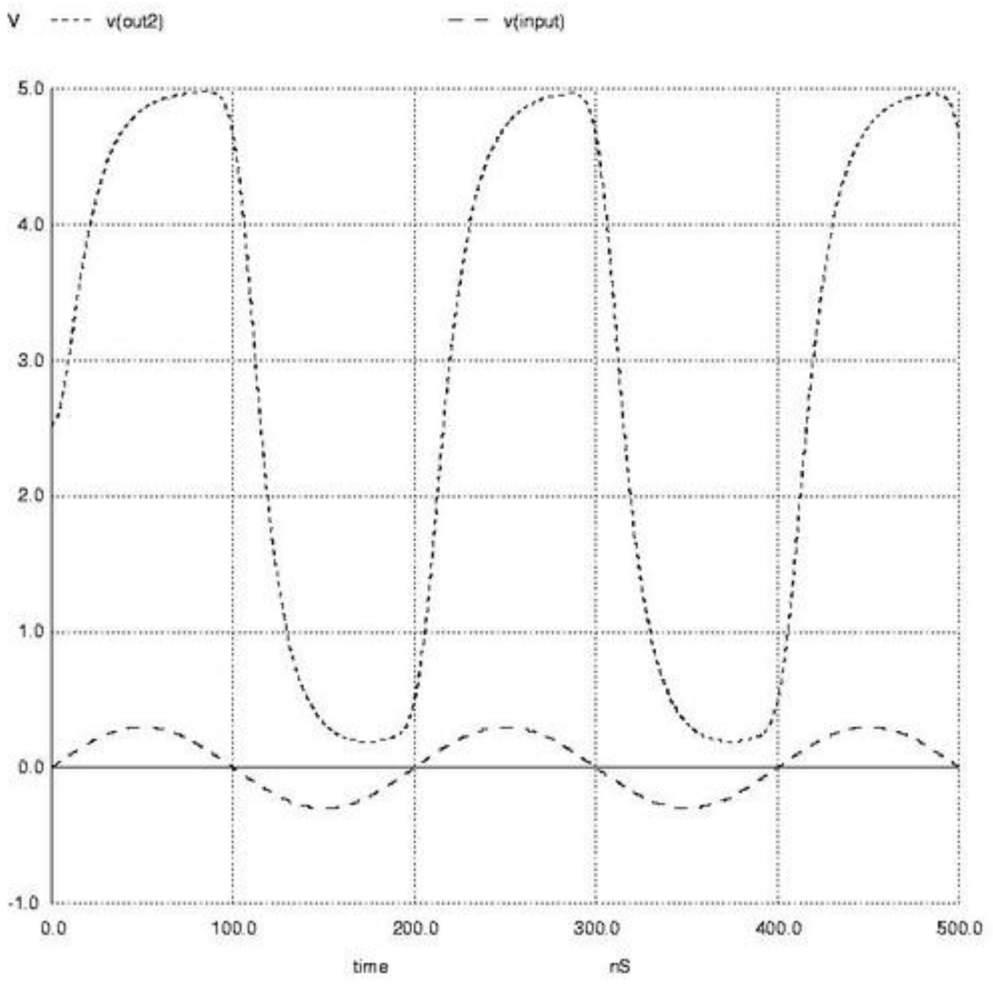


Figure 6: Input and Output Voltage Sine Wave

Summary

This article is far from an exhaustive treatment of what SPICE can do for you and how to use it to its fullest advantage. SPICE is both prevalent and useful to engineers, and has been so for nearly 30 years. If you never use SPICE, I hope you caught at least a glimpse of an engineer's bread and butter. If you use or will use SPICE, I hope this article gave you some insight into its use on Linux machines.

References

Kevin Cosgrove is a design engineer at Tektronix. In his spare time he might be found tinkering with Linux, playing drums, learning the piano, or running the smallest independent record label in town. Comments are welcome at kevinc@doink.com.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Porting Scientific and Engineering Programs to Linux

Charles T. Kelsey, IV

Gary L Masters

Issue #39, July 1997

You can port nuclear engineering programs written in FORTRAN and C to Linux easily and efficiently.

The relatively exorbitant cost of hardware and compilers necessary to put together a fully functional engineering workstation puts them out of reach of most students or professionals, who desire a workstation at home. Even for large companies cutting costs without loss of efficiency is desirable. With the advances in Intel processor speed and the ever growing software base available for Linux, the combination presents a good solution for the low cost workstation.

For engineers and scientists a full featured FORTRAN 77 compiler is a must. Additionally, a readily accessible library of numerical functions and subroutines is needed. Full featured FORTRAN 77 compilers regardless of platform, such as Lahey's compiler for DOS and Sun's compiler for SPARCstations, are priced at around eight-hundred American dollars. Commercial numerical libraries, such as IMSL, can cost thousands of dollars.

Compiling Scientific and Engineering Code Under Linux

Readily available under Linux are a couple of free FORTRAN 77 options. The first is f2c, a FORTRAN to C source code converter. The other is g77, a FORTRAN compiler produced by the GNU project. The limitations of f2c leave g77 as the only viable alternative for compiling large complex applications written in FORTRAN 77 under Linux. Short pieces of FORTRAN code were written as needed for particular engineering tasks and were compiled with g77 to test the compiler abilities as a tool for day to day engineering activities. It was found that g77 produced efficient binaries and behaved as a good FORTRAN compiler should.

The next step was to compile a more significant piece of engineering code that is commonly run on workstations in the ten to twenty thousand dollar range. As radiation safety professionals, a code frequently used is MCNP, that implements the “Monte Carlo” method to transport neutral particle radiations. (The current version of MCNP is 4A; 4B will be released soon.) MCNP represents hundreds of man years of coding and is considered the best available technology for this type of calculations. Like most science and engineering code packages with long development histories, MCNP is written predominantly in FORTRAN 77, and language conversion is not considered a reasonable development step.

One thing that makes porting this code to a new platform somewhat challenging is that it is a safety related, pedigreed code, so one cannot just start changing things. Any changes must be made using patch files with the provided implementation utility. This utility is called PRPR and is also written in FORTRAN 77. PRPR is relatively simple code. The first step in porting MCNP to Linux was to compile PRPR, without modification, using g77. This operation was successful —an encouraging but misleading fact. The rest of the package was not as cooperative.

MCNP had already been ported to numerous platforms, everything from VMS to UNICOS. The first obvious approach was to try the patch files provided for similar platforms. None were successful, but these attempts did result in numerous educational error messages.

An initial hurdle to overcome was the absence of the **fsplit** command in Slackware distributions. This problem was resolved by getting **fsplit** from a BSD distribution.

The second hurdle was the absence of some VAX FORTRAN extensions, specifically those related to signal handling, date and time calls and execution timing. Although not part of the FORTRAN 77 standard, they are included with many of the commercial FORTRAN compilers. The necessary functions of most of the VAX FORTRAN extensions are simple enough to replicate with short C routines that can be linked with the FORTRAN objects when the executable is produced. The MCNP source includes a C routine to handle execution time.

Once the VAX FORTRAN extension problems had been solved, the remaining tens of error messages were related to problems with integrating the FORTRAN and C objects. With MCNP plotting routines written in ANSI C, the integration of objects is necessary to generate a fully functional executable. The errors result from incorrect syntax of subroutine declarations, which varies from one FORTRAN and C compiler pair to another.

Once the executable binary had been produced successfully—that is, a compile completed with no errors—the provided test problems were run. Test problems are provided with safety related codes, so that the user can verify not only that they compiled a binary, but also that the binary produced reasonable answers. A comparison of the test problem output from the Linux binary to the supplied output revealed only differences attributable to differences in architecture. The machine that produced the standard was a SPARCstation 5, and the first Linux box to run the MCNP binary was a 75 MHz Pentium.

Since that first platform, MCNP Version 4A has been complied and run on 100 and 200 MHz Pentiums. While the Pentiums are not quite as fast as a SPARCstation 20/151 (150 MHz HyperSPARC), they are quite fast when their cost is considered relative to the SPARCstation. Note that the referenced HyperSPARC module alone costs over four thousand dollars.

The PRPR implemented patch files necessary to compile MCNP Version 4A under Linux are shown in Listing 1 and 2. Note that both the FORTRAN and the C patch are necessary. It should also be noted that the best MCNP graphical behavior occurs when running under the OpenLook Window manager.

Listing 1. FORTRAN Patch File

Listing 2. C Patch File

While we do not expect that all scientists and engineers use MCNP, we do hope that this documentation of the MCNP port will be helpful to those looking for an inexpensive workstation to run their code.

Library of Numerical Routines for g77 Under Linux

A free collection of routines written in FORTRAN 77 is SLATEC. This collection includes over 1400 mathematical and statistical subroutines that are very well documented with comment lines in a standard format at the head of each piece of source code. In order for this collection to be efficiently used by a work group sharing a workstation, it was found that the creation of a library archive was most appropriate. The **more**, **head**, and **grep** commands with some occasional awking have proven most adequate for finding the subroutine desired and reviewing the documentation integrated in the source code collection.

The SLATEC Version 4.1 library was complied and created using g77 under Linux, and none of the sources required patching. There is one hitch: some of the routines call user supplied subroutines, and therefore, cannot be included in the library. These routines are bvder, bvpor, bvsup, dbvder, dbvpor, dbvsup, dexbvp, drkfab, exbvp and rkfab. The library was created by placing the script in Listing 3 along with all of the **.f** sources in the directory /usr/local/src/slatec and executing the script.

Listing 3

Shell Script to Create the SLATEC Library

The resulting library, libslatec.a, is placed in /usr/local/lib and can be accessed at compile time by any user, assuming /usr/local/lib is in their library path, with a command like:

```
g77 -o somefile somefile.f -lslatec
```

Conclusions

We have demonstrated that with a little work g77 can be used to compile very significant FORTRAN code packages. With the addition of two simple C routines, g77 was used to compile MCNP, and with no source code modification at all, it compiled the vast mathematical source collection know as SLATEC. g77 has proven to be a FORTRAN compiler that can truly make Linux on Intel hardware, a workstation option for the engineer and scientist, especially one on a budget.

More information about Charles' and Gary's computational endeavors is available at www.csn.net/~ckelsey.

Charles Kelsey (ckelsey@devcg.denver.co.us) is a Radiological Engineer at the Rocky Flats Environmental Technology Site. Charles' duties include general health physics, some heavy number crunching and supplemental care and feeding of the department's UNIX workstations. He tries to stay away from VMS. These guys are so weird that they even write nuclear engineering applications for the HP48 calculator.

Gary Masters (gmasters@devcg.denver.co.us) is a Radiological Engineer at the Rocky Flats Environmental Technology Site. Gary's duties include general health physics, some light number crunching and the care and feeding of the department's UNIX workstations and DEC Alpha database server that runs VMS (yuck). Comments are welcome, and subject to being funneled to /dev/null.

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Linux Out of the Real World

Sebastian Kuzminsky

Issue #39, July 1997

Debian Linux has taken flight aboard the Space Shuttle Columbia.

Through the National Aeronautics and Space Administration (NASA), the United States government provides space flight capability to its people; you can rent volume on a Space Shuttle mission and fly a payload into low Earth orbit. Because of the considerable cost involved, in practice, many of the organizations who rent space do so with government grants. One such grant belongs to Bioserve Space Technologies. Bioserve is a sponsored NASA Center for Space Commercialization, operating out of the University of Colorado at Boulder. Here, a group of students (from undergrad through post-doc) and teachers from many engineering disciplines work together to produce payloads that perform various experiments on the Shuttle.

This article describes one such payload, called the Plant Generic Bioprocessing Apparatus (PGBA), and the NASA systems used to communicate with the experiment.

Experiment and Flight Overview

PGBA is a Space Shuttle payload experiment designed to study plant growth and development in microgravity. It flew in the Space Shuttle Columbia, on flight STS-83 on April 4, 1997. The experiment is centered on a small hydroponics plant-growth chamber adapted for use in microgravity. The chamber is fitted with a large number of sensors and actuators, all connected to a 486 PC/104 computer running Linux. This computer monitors and controls a number of environmental conditions within the plant-growth chamber. The data produced is stored locally in the orbiter and transmitted to ground side over an unreliable bidirectional low-bandwidth link provided by NASA. A dedicated ISDN line connects the Marshall Space Flight Center (MSFC) in Huntsville, Alabama with our ground side support equipment in Boulder, Colorado. Here the biologists analyze the data, and we relay it over the Net to the Kennedy Space Center (KSC) in Cape Canaveral, Florida, where a ground-control replica of the experiment mimics the environmental conditions, “on Earth as it is in Heaven.”

Figure 1. The completed BGBA flight unit, just before transfer to NASA.

The plan was to subject the experiment to several relocations within the orbiter after launch. PGBA was to be launched and powered on in the mid-deck. After two days in orbit it was to be moved to the SpaceLab module, where it would be mounted in the Express Rack and connected to the Rack Interface Computer (RIC) that provides both the uplink and the downlink. Two days before landing, it would be disconnected (cutting its communications with ground side) and moved back to the mid-deck. Each of these moves would require astronaut effort (shutting down, moving and bringing the experiment back up) and a loss of power to the experiment. We could have launched and landed right in the Express Rack, but the moving maneuver would allow NASA to test the techniques and hardware that will eventually be used to move experiment payloads between the Space Shuttle and the International Space Station.

Unfortunately, a hardware failure on the orbiter itself forced an early return after less than 4 days in orbit, instead of the planned 16 days. A fuel cell providing electrical power to the orbiter started to fail, and the mission was aborted to minimize risk to the crew. The fuel cell problem was discovered within the first two days in orbit, before PGBA was scheduled to be moved to the Express Rack. Four days in orbit was not enough time for the effects of microgravity on plant growth to manifest themselves, and from a science standpoint the experiment was considered a complete loss. However, it was not without value, since we now have a flight-tested and known working experiment. NASA is eager to test the Station transfer procedure, and the scientists are eager to get their data. A repeat flight has been tentatively scheduled for early July, 1997—same crew, same vehicle, same payloads, just a new tank of fuel.

Figure 2. Closeup of the inside of the unit. The little box on the bottom right is the computer

I will describe the payload we designed and the mission we originally planned (the same one we are expecting to complete in July) rather than the aborted mission that we actually flew.

Design

How do you design a computer system to handle this situation? Clearly it is a mission-critical item. If the computer fails, the experiment is lost.

Astronaut time is an incredibly expensive commodity. This has two implications: it is desirable to automate normal operation of the payload as much as possible and not to require maintenance or repair in orbit.

The computer system must operate autonomously for the duration of the mission (on the order of two or three weeks). During this time it monitors and controls the conditions inside the growth chamber, using an array of specialized sensors and actuators. It must also communicate with ground side, both accepting input and providing output. Physically, the computer must occupy a small volume.

Figure 3. The light box, containing the fluorescent lights, their fans and the two tiny cameras.

Data produced before the move to the Express Rack and after the move back to the mid-deck would need to be buffered on non-volatile storage. Just before the move to the Express Rack and just before we get the payload back after launch, we would need to buffer a maximum two days worth of data.

The solution we decided on is a PC/104 computer running Linux. PC/104 is an “embeddable” (90 by 96 mm, low power consumption) implementation of the common PC/AT architecture. PC/104 hardware is software compatible with ISA hardware, but the connectors and layout are different. This has obvious advantages: all the software that runs on vanilla desktop PCs runs unmodified on PC/104 computers. (Incidentally, the PC/104 Consortium just announced the PC/104-Plus spec, which describes an extension to the regular PC/104 architecture that is software compatible with PCI. For more information on the PC/104 standard, see http://www.controlled.com/pc104/consp1.html.)

Figure 4. The plant-growth chamber

We chose Linux for “soft” reasons. The job could be done in MS Windows, on a microcontroller or on a Turing machine, but who would want to? The tools and computing environment available to programmers in the more advanced operating systems make life so much nicer.

Last year on STS-77 we flew two payloads with similar computer systems. This year we used Linux, last year we used DOS. The DOS software worked and was functionally almost equivalent to the Linux version. Notably, it lacked image capture, downlink of images and local storage of data logs. It was switched because the DOS version was monolithic, more difficult to understand, debug and expand, and it was difficult to reuse the code.

The Computer

The motherboard is a CoreModule/4DXi from Ampro with an Intel 486 DX4 100 MHz CPU, an IDE and a floppy controller, two serial ports, a parallel port and a hardware watchdog. The 4DXi ships with 4MB RAM that we upgraded to 16MB. Ampro hardware has in our experience been consistently reliable, well documented and given excellent support.

Figure 5. The flight unit is on the left, the ground control on the right

The payload needs three serial ports: one to talk to the Rack Interface Computer (that provides the uplink and downlink), one to talk to the astronauts (through a touch-screen) and one for connecting a terminal for development on the ground and for resolving any emergencies that may crop up in space. We needed one serial port in addition to the two on the motherboard, so we added an MPC302 card from Micro/sys that provides two additional serial ports and a second parallel port. The MPC302 card supports shared IRQs (interrupt requests)—a big win.

The touch-screen is a GTC-100 from DesignTech Engineering. It is a touch-sensitive LCD screen with a serial port. It accepts high-level text and graphics commands and reports the location of screen presses. Through this device we provide the interested astronaut with detailed information about the experiment, and a menu interface for control and meta-control.

Figure 6. The chief scientist is installing his babies into the root tray

The experiment is monitored and controlled by a number of bizarre gadgets: accelerometers and gas chromatographs, volumetric pumps and porous condensation plates—your regular Sci-Fi gardening tools. These are in turn monitored and controlled by a number of analog and digital inputs and outputs to and from the Linux box. We are using three I/O cards from Diamond Systems: two “Diamond-MM” for analog I/O and one “Onyx-MM” for digital I/O. These cards provide all the I/O required to perform the process automation and monitoring.

In addition to the numerical data gathered, we are taking periodic pictures of the plants with two miniature video cameras. The cameras are mounted in the “ceiling” of the plant-growth chamber (the side with the lights), and their combined field of view covers the entire “floor” of the chamber (where the plants are). The NTSC video signals feed to an ANDI-FG board from Ajeco. The ANDI-FG has a 3-input frame grabber, a Motorola 56001 DSP and a megabyte of on-board memory. On request, the ANDI-FG delivers to the host CPU a high-quality JPEG-compressed image. Ajeco has been most helpful, providing a Linux driver and excellent technical support.

Plugged in to the IDE controller we have a FlashDrive solid-state disk from Sandisk. We chose to go with a solid-state disk as opposed to regular rotating magnetic media, because our system needs to operate under heavy vibration for extended periods of time. The FlashDrives are more expensive and have low capacity, but they are guaranteed to operate under 1000 G shock and sustained 15 G vibration without damage. We have plenty of persistent storage, although we could easily increase that to several hundred megabytes should we need it by using larger FlashDrives. 40MB is enough disk space for the software we need, plus enough to buffer 5 days' worth of data and images. A normal, successful mission would need only two days' worth, but having the extra space made sense.

Figure 7. The plants on the launch pad

My only complaint about this hardware is that most PC/104 cards (all the cards listed above except for the CoreModule) provide only an 8-bit bus, thereby allowing only the use of IRQs 2-7. The CoreModule, being 16-bit, supports the full range of IRQs. Between our I/O cards and serial ports, we are running out of hardware interrupts.

Absent from the above list of hardware is a video card and a network card. In its production configuration, we run the PC/104 without either of these cards. During ground development when we have physical access to the computer, we use a simple serial terminal for a display, and PPP over a null modem at 115 Kbps for networking.

The ground control uses an Ampro MiniModule/Ethernet-II card, a 16-bit Ethernet interface card based on the SMC 9194. The ground control is on the Net from behind NASA's firewall at Kennedy Space Center and gets data from our ground side support computer in Boulder.

Figure 8. The plants on the launch pad

As to software, the experiment is running a customized installation of the feature-rich Debian 1.2 base, plus a few select additional packages (notably a decent editor). We use version 2.0.27 of the Linux kernel, plus Miquel van Smoorenburg's serial-console patch and a couple of nonstandard drivers we wrote ourselves for the analog and digital I/O cards. The manufacturer-supplied driver for the Adjeco frame grabber is a user-space-only implementation. Last but not least, we have the custom automation/communication software suite.

Communications: A High-level Overview

The data produced by the payload is sent over a serial connection to the Rack Interface Computer. It bounces around on the orbiter for a little while, then is beamed to ground side via a satellite in NASA's Tracking and Date Relay Satellite System (TDRSS, everyone calls them Tetris satellites). The data goes through some other NASA systems (including communications-relay vessels in the Pacific Ocean) and, finally, makes it to MSFC. At MSFC, it enters a machine named (in good NASA style) the “Virtual Remote Users Gateway”, VRUG for short. The VRUG is connected via a dedicated NASCOM line to our Remote Payload Operations Command Center (Remote POCC) in Boulder. The data then goes into a pile of ISDN-to-Ethernet routing hardware and into a network card in our ground side support computer (another Linux machine, used for development of the experiment's software and the analysis of returned data). On its screen, the ground side support computer displays squiggly lines (which the biologists like to look at) and pictures of plants. Another channel going up to orbit from ground side exists using the same hardware interfaces (RIC and VRUG).

Figure 9. The plants at the end of the mission

The data from the payload describes the conditions in the orbiter. From Boulder, it is sent over the Net to the ground control experiment in Florida. The ground control is similar to the payload in orbit, but it has an Ethernet card instead of the third serial port and a fragile (but cheap) and spacious magnetic hard disk of the garden variety. The ground control produces data of the same form as the orbiting experiment, but with (hopefully) different content. This data is sent back over the Net to the support machine in Boulder for analysis.

An unusual instance of a common problem affects communications with the orbiter. Each TDRSS satellite can see a small portion of the sky: when the limb of the Earth passes between the orbiter and the satellite, line of sight is lost and no data can be sent between them. Several TDRSS satellites are in orbit, and large portions of the orbiter's possible locations are covered, but not all. When no satellites are visible from the orbiter, no communication is possible. This situation is called a Loss Of Signal (LOS). NASA announces the LOSs with high accuracy and long precognition, but they still cause headaches for experimenters. (E-mail your politicians and ask for more Tetris satellites.)

Figure 10. The plants at the end of the mission

NASA does not guarantee the delivery or correctness of data sent through their pipes. I once asked a member of their technical staff how reliable the channel is, and he replied “Oh, I think probably no more than one corrupt or dropped character in a hundred.” Observations made during last year's experiment indicate that the error rate is significantly lower than that estimate.

Communications: NASA's Interfaces

When you rent volume on the orbiter for your experiment, you can also rent bandwidth to ground side. You must specify the number of bits per second to reserve for your payload, and you are guaranteed no less. You then get a connection to the Rack Interface Computer. The RIC presents a three-wire RS-232 connection: transmit and receive only, no handshaking.

Data generated by the experiment must be encapsulated in little packets in accordance with a specification from NASA. The fields in the header and footer of these packets are used for routing within the orbiter's communication equipment and include a checksum. If the RIC accepts your packet, NASA will do their best to deliver it to your machine at ground side, but no guarantees are made. Data sent back to the payload from ground side is encapsulated in the same packets and go over the same wires. All packets traded between the payload and the RIC contain data that is from or to the ground side support computer, wrapped in RIC packets.

There is no change at the RIC interface, no automated notification from NASA to the payload that a LOS is imminent or occurring. The obvious way to do this notification would be to use the regular RS-232 handshaking lines.

Our Remote POCC is in Boulder, Colorado. At this end of the line, NASA presents a twisted pair Ethernet interface. You connect using TCP/IP to two specified ports on two specified hosts on this network. These computers are collectively called the Virtual Remote Users Gateway (VRUG). The VRUG interface is more complex than the RIC interface.

All communication with the VRUG (in both directions) is encrypted. The computer people at MSFC asked us to identify our operating system, and then supplied us with two object files, containing compiled C-callable functions to encrypt and decrypt data. The data sent over the TCP stream between our computer and theirs is packetized, but using packets different from those used by the RIC. These packets can contain data to and from the orbiter, commands to configure the VRUG interface and “telemetry” data from the orbiter (a standard data set provided by NASA at no extra charge, describing ambient conditions within the orbiter). Checksums are dutifully computed and checked on data going over the TCP link.

Communications: Some Software Design Thoughts

Two pairs of processes want to communicate between the orbiting experiment and the ground side support computer. The main control process in the experiment communicates with a recorder/data display/remote control program on the support computer, using several different types of data (events-log, sensor readings, and control-parameter settings going down; control and meta-control commands going up). Another pair of processes mirror directories from the payload to ground side, in order to send data that was buffered before communications were established. Images from the frame grabber are sent using this method.

In this situation, it is natural to want a packet-switching, multiplexing communication system with an interface available to many processes. Since the channel is unreliable, you want some type of validation of received data. The usual networking code is not usable, since there are no packet-drivers for the interfaces NASA presents. In the interest of simplicity and code reusability, we chose to implement a modular user-space communication system.

We wanted the communications interface presented to our communicating processes to be identical on both machines (experiment and ground side support computer). Since these two machines need to talk to different hardware and software interfaces, we abstracted the NASA interfaces from the multiplexer. Between the multiplexer and NASA sits a process that performs the packetizing and unpacketizing they want us to do (possibly fragmenting and defragmenting multiplexer packets) and relays the data. The end result is that the payload and the ground side support computer can communicate in a UDP-like fashion. Multiplexer packets sent are guaranteed to arrive intact and correct or not at all. It's a miniature networking stack in user land.

When the old DOS version of the payload flew in 1996, we rented space in SpaceHab. SpaceHab is a privately owned company that rents a large volume in the Shuttle payload bay and some services (power, communications, etc.) from NASA, and then turns around and rents smaller quantities of volume and services to experimenters. The economic relationship between the three parties (NASA, SpaceHab, experimenter) in this situation defy comprehension by the author. Anyway, SpaceHab provides a significantly more functional communications interface, called the Serial Converter Unit (SCU). Sure, it's still a 9600 bps serial line, but the SCU has (angels sing) flow control.

Sebastian Kuzminsky is an undergraduate in Computer Science and Applied Mathematics at the University at Colorado at Boulder. If space flight work were not so fun and time consuming, he would have been a graduate by now. Questions about PGBA and other Bioserve payloads are welcome. He can be reached via e-mail at sebastian.kuzminsky@colorado.edu.

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Octave: A Free, High-Level Language for Mathematics

Malcolm Murphy

Issue #39, July 1997

A quick look at a language designed to manipulate matrices and provide other numerical functions.

For numerical computing, high level languages offer advantages over more traditional languages, such as FORTRAN or C. Built-in graphics capabilities, automatic variable typing and flexible data structures combine to provide an environment in which it is easy to develop your ideas without having to fight with the language. That's not to say that FORTRAN and C are of no use, just that sometimes you want to make life a bit easier.

Matlab is a one such language. It is available on many platforms (including Linux) and provides powerful facilities for manipulating matrices, as well as other numerical functions. Unfortunately, Matlab is commercial software and wasn't available for Linux until recently (in the last twelve months or so). However, there are other, freely-available alternatives, and Octave is one such alternative.

Superficially, Octave looks very much like Matlab, and the description in its LSM entry reads “GNU Matlab—A numerical matrix mathematics program.” To begin, type **octave** at the shell prompt, and Octave greets you with its own prompt. Now we can start doing math.

Matrix Operations

As you might expect, entering and manipulating matrices is one of Octave's strengths. (In order to differentiate Octave commands from the output, the prompt **octave:*number of the command*** precedes the commands in the examples below.) We can enter a matrix with the command:

```
octave:1 a=[1 2 ; 3 4]
```

Octave then reports the result of the command, namely:

```
a =
        1  2
        3  4
```

To suppress output, simply place a semicolon after the command. Note that we didn't have to worry about declaring the size or type of the matrix **a**, we just type it in and start working with it. For example, we can get the transpose of **a** by typing:

```
octave:2 a'
      ans =
        1  3
        2  4
```

Arithmetic operators such as **+**, **-** and ***** act as matrix operators unless they are preceded by a period, in which case they act in an element by element sense. Octave also provides the forward and backward slash operators that perform matrix right and left division. Again, these can be used in an element-wise fashion. So, given another matrix:

```
octave:3 b=[1 0; 3 2]
b =
  1  0
  3  2
```

we can find the matrix product of **a** and **b** with the command:

```
octave:4 a*b
ans =
   7   4
  15   8
```

Or define the (**i,j**)th entry of the product is the product of the (**i,j**) entries in **a** and **b** with the command:

```
octave:5 a.*b
ans =
  1  0
  9  8
```

Matrix elements can easily be selected by index, so to get the (1,1) entry of **a**, type:

```
octave:6 a(1,1)
ans=1
```

We can select a row or column using the colon operator, exactly as in Matlab. Thus, to select the first column of **a** type the command:

```
octave:7 a(:,1)
ans =
  1
  3
```

And to select the first row of **a** type:

```
octave:8 a(1,:)
ans =
  1  2
```

As well as these elementary operations, Octave provides functions that perform higher-level operations, such as finding the eigenvalues of a matrix, by using a command like the following:

```
octave:9 eig(a)
ans =
   5.37228
  -0.37228
```

Alternatively, you can find the eigenvalues and eigenvectors by giving the following command:

```
octave:10 [v,d]=eig(a)
v =
   0.41597  -0.82456
   0.90938   0.56577
d =
   5.37228   0.00000
   0.00000  -0.37228
```

This is a demonstration of one of the main advantages of using a high level language like Octave. Writing a FORTRAN or C program to find the eigenvalues of a matrix would take a lot more time and effort than it does in Octave. Moreover, Octave routines are usually based on well-known, high quality algorithms, so you can have faith in the results.

Octave provides many other matrix routines, which are detailed in the manual and in the on-line help system.

User-Defined Functions

Octave also lets the user define his own functions via the **function** keyword. A function definition looks like this:

```
function [output values] = name (input values)
   sequence of commands
endfunction
```

The input and output values are optional, so it is possible to write a function that takes no arguments and returns no values, such as

```
octave:11 function hello
   printf("hello\n")
endfunction
```

The **printf** statement prints the quoted string to the screen, and the \n is interpreted as a newline. Invoke the function by typing its name:

```
octave:12 hello
   hello
```

Obviously, functions that don't take arguments or return values aren't all that useful. To accept arguments, list them after the function name in the following manner:

```
octave:13 function add(x,y)
   x+y
endfunction
octave:14 add(1,2)
   ans = 3
```

The output came from the statement **x+y**--if we had ended the line with a semicolon, there wouldn't have been any output from the **add** command. To assign the output to a variable, define the function in this way:

```
octave:15 function sum=add(x,y)
   sum=x+y;
endfunction
```

Now, if we type **add(1,2)**, we get exactly the same result as before. However, by defining an output variable, we can assign the result of the **add** function to a variable in this way:

```
octave:16 fred=add(1,2)
   fred=3
```

A very powerful feature of Octave is the ability to return multiple values from a function. This feature exists in Matlab, but not in FORTRAN. For example, the following function:

```
octave:17 function [sum,diff]=sumdiff(x,y)
   sum=x+y;
   diff=x-y;
endfunction
```

returns multiple values when invoked as:

```
octave:18 ns,d]=sumdiff(1,2)
   s = 3
   d = -1
```

Functions can be defined at the keyboard, as we did in these examples, or stored in a file and used again. This lets you build a suite of routines for whatever tasks you want. All you have to do is put the files (identified with a **.m** suffix) somewhere where Octave can find them. The built-in variable **LOADPATH** specifies where Octave should look for the **.m** files. Many of Octave's standard functions are defined in **.m** files. You can also access user-supplied C++ routines within Octave, although this feature is not yet fully developed.

Octave also provides a full programming language, with flow control and looping constructs, as well as extensive input-output facilities. It is possible to write quite sophisticated programs in Octave, and development time is considerably shorter than you would expect using FORTRAN or C.

Other Capabilities

Although Octave has very strong matrix capabilities, it has many other features as well. For example, it has routines to manipulate polynomials. A polynomial is entered as a vector of coefficients; so, for example, the polynomial x3+3x2+2x-1 can be represented by the vector:

```
octave:19 mypoly=[1 3 2 -1]
   mypoly =
     1   3   2  -1
```

We can then differentiate **mypoly** using the command:

```
octave:20 polyderiv(mypoly)
   ans =
     3  6  2
```

or integrate it by:

```
octave:21 polyinteg(mypoly)
  ans =
  0.25000   1.00000   1.00000  -1.00000   0.00000
```

Note that Octave uses zero as the constant of integration. We can also evaluate the polynomial at a given value; thus, to find the value of **mypoly(2)** use the command:

```
octave:22 polyval(mypoly,2)
   ans = 23
```

If we want the roots of the polynomial, use:

```
octave:23 roots(mypoly)
   ans =
     -1.66236 + 0.56228i
     -1.66236 - 0.56228i
      0.32472 + 0.00000i
```

Note that Octave is quite happy with complex numbers, even though all the examples I've given have been real. There are also routines to convolve and deconvolve polynomials, form companion matrices and characteristic polynomials, and form a partial fraction representation of the quotient of two matrices.

Other features include functions to solve systems of nonlinear equations, solve differential and differential-algebraic equations, perform quadrature and collocation, as well as statistics, control theory, signal processing, image processing and optimization routines. The manual indicates areas where the developers hope to extend Octave's capabilities.

Graphics

Octave provides graphics capabilities via the Gnuplot program, which has to be obtained separately. The advantage of this is that Octave supports all the output devices Gnuplot supports, including the Linux terminal, which might be of interest if you have a low-memory system.

Octave provides two low-level graphics functions, **gplot** and **gsplot**, that behave almost exactly like the Gnuplot functions **plot** and **splot**, and also provides several higher-level plotting functions based on the graphics functions found in Matlab 3.5. Two and three-dimensional plotting commands are also available.

If you are familiar with Gnuplot, then you will probably appreciate the flexibility offered by access to Gnuplot's commands. However, the higher-level commands offered by Octave are very easy to use and you may find you don't have to use the Gnuplot commands at all.

Conclusions

Octave is a flexible, powerful, easy-to-use, high-level language designed for numerical computations. It comes with a very readable 200+ page user manual, and a help system based around the GNU info system. The main advantage of a high-level language over a language like FORTRAN is that development time is usually considerably shorter using a high-level language. This allows for easy prototyping and experimentation.

Although the documentation doesn't claim that Octave is intended to be a Matlab clone, or Matlab compatible, Octave is probably the most Matlab-like of the freely available high level languages. It's not exactly the same, but I was able to convert a suite of Matlab m-files that perform finite element analysis of the Navier-Stokes equations to Octave very easily.

Octave has many more features than I have described here, but I've provided an overview of its main strengths. If you're looking for a language for numerical work, Octave is certainly an option. I don't think you can directly compare Octave with such languages as RLaB, SciLab and Yorick—they all do different things, and which you choose depends on what you want to do as well as personal preference. My preference is Octave.

Malcolm Murphy still wishes that he had discovered jazz before he gave up clarinet lessons at an early age. He considers himself too old (or too lazy) to start again now, so he plays the guitar instead. If you have an uncontrollable urge to send him some e-mail, his address is Malcolm.Murphy@bristol.ac.uk.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Programming with the XForms Library

Thor Sigvaldason

Issue #39, July 1997

The XForms home page calls XForms “a GUI toolkit based on Xlib for the X Window System. It features a rich set of objects, such as buttons, sliders, and menus, etc., integrated into an easy and efficient object/event callback execution model that allows fast and easy construction of X applications.” With this first of three articles on XForms, you can ease into programming for X without having to know exactly what “object/event callback execution” involves.

Where did Linux come from? That's a good question, and one that seems to confuse a lot of new users. The short answer, of course, is that Linux was written by Linus Torvalds (the original and central force behind the Linux kernel). But that cursory response misses much of the spirit of Linux. When you go into a computer store and buy a Linux CD-ROM set, you're getting a copy of the intellectual output of literally thousands of programmers. Most of these people have never met each other. Thanks to the Internet, however, this geographically dispersed group has managed to create a formidable array of tools and applications. Linux works so well that it creates the illusion that there is some kind of organization and control structure in place—some central authority directing every aspect of its development. This makes it easy to forget that every application you run—from an ASCII text editor to a Z-Modem download—exists because somebody, somewhere, sat down one day and said, “I can't find a way to do this under Linux, so maybe I should try to write my own program.”

This article is the first in a three-part series designed to introduce the reader to a programming tool called The XForms Library. As its name suggests, XForms is a set of tools and routines that can be used to easily create programs that run under the X Window System. The overall goal is to take a reader who has never written an X-based piece of software to the point where he or she can start writing their own applications and utilities. The series assumes the reader is familiar with the C programming language, since XForms is C-based. If you don't yet know anything about C programming, now may be a good time to take the plunge. Buy a book or two and start fiddling with it. With a few weeks of study, you'll probably have enough C under your belt to be able to follow the series by the time the next article appears. It is also assumed you have a working C compiler and the X Window System is already installed.

It's the author's hope that readers will eventually be able to contribute an XForms-based program to Linux. The next time you post a “where can I find a program to do x” message to comp.os.linux.misc and don't get an answer, think about whether it's something you might be able to write yourself. You probably won't earn a cent for your troubles, but the process is not without its rewards. One day, a few months after uploading your package to Sunsite, you may find yourself in a software shop that has just received a new shipment of Linux CD-ROMs. Up on a shelf, probably squeezed between the latest commercial software from Microsoft and Corel, will be a few kilobytes of code you wrote and which people all over the world are using.

Outline of the Series

The purpose of this first article is to explain how to get and install XForms. We also take a first glance at the process of creating an XForms application and actually write a couple of simple programs.

The next two articles, which will appear in the August and September issues of *Linux Journal*, will expand on this process. They will go through all the steps required to create useful software with XForms with the aid of an example application, a simple game theory simulator. Don't worry if you have no idea what game theory is, since all you need to know about it will be explained in the process. The choice of programming example is driven by the author's interests, and is irrelevant to the central purpose of the series: learning to use XForms.

Details on where to get all of the software mentioned are available in the Resources section at the end of this article. There is also a web page for this series which can be found at http://a42.com/~thor/xforms/. This site includes links to all relevant software, as well as listings of all the source code used in the series.

An Overview of XForms

The XForms library was written by T. C. Zhao and Mark Overmars. It is free for non-commercial use, but if you're planning on selling your application, you will need to set up a monetary licensing arrangement with the XForms authors. If you're a free software “purist”, you may want to think about using V instead of XForms (see the next section).

The most important parts of the XForms package are the actual library (libforms) and the forms.h include file. The former contains functions for creating buttons, menus, and so on. The include file declares these library functions, which makes them available for use in your software. If you're a relative newcomer to all this, don't worry—we'll be going through every step required to get things working.

There is also a very thorough manual available in HTML format, which explains all of the available routines. A large set of example applications, which are referred to in the documentation, are included in the main distribution. These examples are very useful for getting good ideas on how to implement various procedures.

Finally, there is a program called fdesign included with XForms that can be used to design user interfaces. This is a real time saver, since you can use your mouse to control the placement of buttons, menus and other objects. Your design can then be saved to a file and easily brought into your source code. Complicated sets of overlapping windows with large numbers of graphical elements can thus be created in a relatively short period of time.

Since XForms is actually built around the X11 library, any program you write is highly portable to other X-capable systems. Although readers of this magazine will probably be developing for Linux, it's nice to know users of FreeBSD, Sun and other systems will be able to run your application if they want.

Alternatives

Since learning any programming library involves a substantial investment of time, it may be worth your while to consider a few alternatives before jumping headlong into XForms.

The following discussion is far from exhaustive and is based solely on the idiosyncratic experiences of the author.

Motif is a very common library which is used to build many commercial X-based programs. To create your own Motif applications, you need the Motif development library, which is fairly expensive (at least by Linux standards). The main advantage to Motif is that it is a mature system which can create visually appealing applications. The main drawback is its price. There is a free version of Motif in the works (called LessTif), but it is not yet 100% functional.

Another library with a licensing arrangement similar to XForms is called Qt. The Qt library is free for non-commercial use under X, but is also available for Microsoft Windows and other operating systems. This has the great advantage that if you write a program under Linux, people can also use it on completely different operating systems. It is also the core of the new KDE Desktop Environment, which should be available by the time this article appears in print. If KDE catches on, then Qt will become very popular in the Linux world. The Qt library is based around C++ (rather than plain old C, which XForms uses).

If you want an X development system that is truly free, then you may want to look at V. This is another C++-based library, which includes all the standard features, such as buttons, menus, etc. As the author of V admits, it is probably not suitable for trying to write a state-of-the-art interface, but it is a reasonably complete package, and it's difficult to argue with the price.

Finding and Installing XForms

The main task in installing XForms is to get the library and include file into the right places on your system. The Web and ftp sites for XForms are listed at the end of this article. Most people will want to grab the ELF version, but if you are still running a.out, then make sure you get the relevant file. The distribution includes the library, more than 50 example applications, and the fdesign program. You should also get the reference manual, which is available in a separate file.

At the time of writing, the most recent version of XForms is 0.81, and the Linux package is in a compressed tar file of roughly 600KB (i.e., it has a .tgz extension). You should make sure you are logged on as root and unpack the distribution somewhere like /usr/local/. The archive installs into a directory called xforms. It's generally a good idea to check the Readme file included in the distribution for advice on installation.

If you have a more or less standard setup, then all you need to do is go to the xforms directory and type **make install**. This should copy the library and include files into the right places. If this doesn't work (or you want to do things by hand), just make sure forms.h ends up somewhere gcc can find it (i.e., in /usr/include/) and all libform files end up in the same place as libX11 (i.e., in /usr/X11R6/lib/).

If you want to verify things are installed correctly, change to the DEMOS subdirectory and try **make demo**. If this does not work, go back and ensure the library and includes are where they're supposed to be.

Your First XForms Program

With the library and include file installed, it's time to write your first program. It is a C tradition that a first attempt should always say “hello world”, so we'll write one that does just that. Either type in the source code for xhello.c (see Listing 1) or copy it from the series web site.

Listing 1. xhello.c

As the source code suggests, you should be able to compile the program with the command:

```
gcc -lX11 -lforms -lm xhello.c -o xhello
```

The command **xhello** should start up the program (make sure you're running X). You may have to type **./xhello**, depending on how your shell's PATH variable is set. The running program should look something like Figure 1. Note, you must include the linking options for this to work. In particular, **-lX11** links your code to the standard X library, which XForms depends on (libX11 is included in all X distributions). The forms library is linked in with **-lforms**, and it uses some math routines which must be linked with **-lm**.

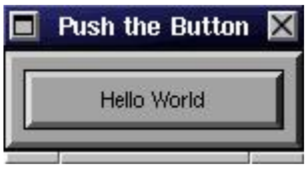


Figure 1. Hello World

Although it doesn't do very much, the xhello program shows the basic steps involved in writing an XForms application. First, we include the forms header file, which gives us access to the XForms routines. Then, **fl_initialize()** is called to let XForms set itself up. We pass all command line options (stored in **argv**) to this function, so that XForms can pull out those it wants. For example, our xhello program already understands command options like **-share** (to share the colour map) and **-display** (to open itself on a specified X display).

With initialization taken care of, we can create as many graphic elements as we like. In this example, we have just one button located in a single window.5 We then show the window we've created with the **fl_show_form()** function. To get the program to wait for the button to be pushed before exiting, we invoke **fl_do_forms()** which waits until the state of our window changes before returning.

5. This brings up a note on terminology. Many people think of a rectangular collection of objects on a screen as a window. The XForms library refers to just such a collection as a form. For the rest of this series, it is best to consider the two terms interchangeable.

There are a couple of things to notice at this point. First, if you have never written an X application before, then congratulations are in order (give yourself a pat on the back). Second, although installing the library may have taken some time, the result was well worth it. With just two variables and well under a dozen function calls, we have a fully fledged running program. Not only that, our program is attractive: the colour scheme is easy on the eyes, the button is nicely shaded to appear three dimensional, etc. Finally, the program is quick and responsive, even on slow hardware. This is due to a combination of factors, the most important of which are that XForms is quite efficient, and everything has been done in pure C.

A Little More Detail

If you're already thinking ahead to greater things, you are probably a little confused about how to get an XForms application to do anything other than return after a single button is pushed. The next programming example provides a hint, but the whole story will have to wait for the next two articles in the series.

The basic idea is to proceed as in the xhello example, but to add some functionality to each action. We do this by creating a multi-lingual “Hello World” program called xmulti. The source is shown in Listing 2, and is also available on the series web page. Save this file as xmulti.c. It should compile with the command:

```
gcc -lX11 -lforms -lm xmulti.c -o xmulti
```

Listing 2. xmulti.c

The program can now be executed by typing **xmulti**, and should look like the example shown in Figure 2. Examination of the xmulti source code reveals the fundamental steps involved in creating an XForms program are as follows:

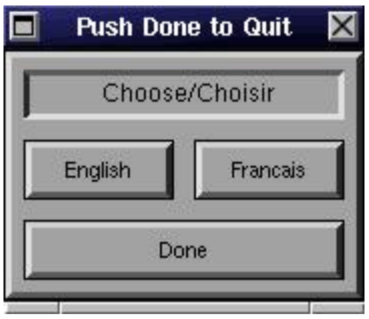


Figure 2. Mutlilingual Hello World

1. Include forms.h to access the XForms routines
2. Call **fl_initialize()** as soon as possible
3. Set up your graphical interface by creating forms
4. Assign actions to relevant objects by setting callbacks
5. Show one or more forms
6. Turn control over to **fl_do_forms()**

The only thing new here is point 4, which our original xhello program did not include. In xmulti, the English and French buttons are set to call the routine **set_language()**, which changes the display. But the basic idea is very general, and you can easily add buttons, menus, etc., that call complicated functions, display other forms, or what have you.

Coming Next Month

Next month, we'll expand on this basic discussion by writing a more complicated program. This will involve using menus, multiple windows, and a few other refinements. By the time we've done the third article, you should be well on your way to creating useful applications.

If you can't wait to learn more, then you may want to start reading through the XForms manual. Browsing the example applications' source code is also an excellent way to familiarize yourself with the XForms way of doing things.

Resources

Thor Sigvaldason is the author of the statistics program xldlas, which uses the XForms library (see *Linux Journal*, Issue 34, February 1997). He is trying to complete a PhD in economics, and can be reached at thor@netcom.ca.

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Send Your Smile by E-mail

Frank Pilhofer

Issue #39, July 1997

Using UUDeview can relieve you of the problems associated with sending and receiving binary e-mail.

When did you last send or receive an e-mail? Right. The WWW is fun to surf, but the true killer application on the Internet is e-mail. Another similarly popular service is the Usenet, where you can directly address a desired target audience. There is no doubt that these two services are the most personal.

Suppose you want to smile at your pals? Or that you're red with anger? We all know the “emot-icons” to use here. However, you will see no emot-icons on the Web; instead you will see icon smilies or maybe even a photograph of a smile. These same symbols would be more effective in e-mail than emot-icons, and a threatening roll of thunder could really frighten your reader. “Now, that's impossible,” you'll say—and you're almost right. But why?

A Bit of History

E-mail service was conceived in the infant days of the Internet, and the Usenet actually started as an off-line network, where messages were distributed via slow modem lines. At that time, transferring large amounts of data like graphics or audio messages was inconceivable, and both systems were designed to handle only plain text messages.

When the available bandwidth increased, people realized that exchange of binary data instead of just ASCII messages should be possible. Now the design limitations became obvious. But instead of going back to the drawing board with their systems, an algorithm was introduced to *encode* binary data into plain-text messages, thus bypassing the problem—*uuencoding* was born. Using this algorithm, three arbitrary bytes were encoded in four printable characters which could then be safely transported with the old software. On the other end, the recipient could decode the data back into its original form.

Unfortunately, the original uuencoding algorithm wasn't perfect, and several other encodings were launched. First *xxencoding*, then a slightly revised uuencoding, the BinHex encoding, which also took care of both the special characteristics of Macintosh files and the slightly better-compressing *ship* and *btoa* encodings. The chaos was complete—you needed different tools for each of them.

MIME is Introduced

Finally, MIME (Multipurpose Internet Mail Extensions) came into being. This standard enabled the effortless transportation of binary data by introducing Base64, another encoding algorithm. It also addressed other issues such as content presentation and the required localized character set for international messages. With MIME encapsulation, a combination of text, audio and video can be sent in a single message along with instructions (for example, whether the audio is to be played along with the video or afterwards). Using these capabilities, you can add graphic icons to your text and underline the message with deafening thunder. You can also send your message as HTML with links to your home page, or as a fully formatted PostScript document. MIME can significantly update the look of e-mail and the Usenet, making the Internet a happier place.

But life just isn't that simple. Although the MIME standard was first published in 1992, it hasn't been widely accepted despite its many advantages. Some small improvements have made the jump into real life (such as the *Content-Type* header), but the full capabilities have yet to be exploited. Still, most binary data posted on the Usenet is encoded using uuencoding, and some Windows-based email software uses BinHex encoding by default. If you are still using the mail and Usenet software that has served you so well over the years, you are probably stuck with decoding the Base64 data used in MIME messages as well.

Until all users have switched to MIME-compliant software, you must save the messages from your mail or Usenet reader into a file and dissect that file with various tools in order to extract the images and audio clips. And even then it is amazing how much can go wrong.

Enter UUDeview

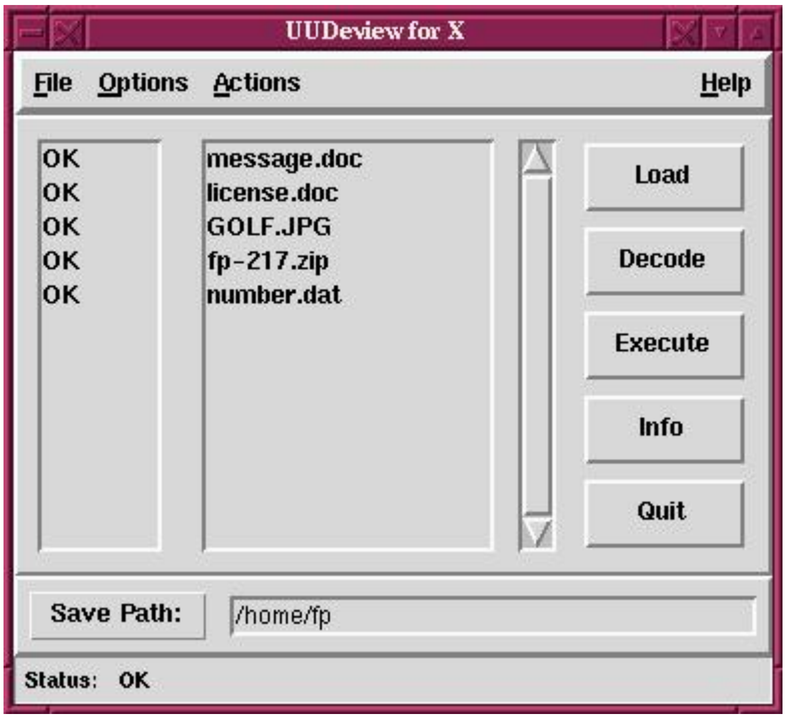
After being disappointed by many similar tools, I developed my own “Smart Decoder” to address the various problems I encounter when receiving an encoded file.

The resultant decoder is a tool called UUDeview. Despite its early development stage—its version number is 0.5—it does its job well. It can read plain encoded data and whole message folders, which are automatically sorted, grouped and threaded in case any data is spread over more than one message (the dreaded multi-part messages). The user can instruct the program to decode each file or not. UUDeview also handles uuencoding, MIME's Base64 and BinHex.

The UUDeview package comes complete with UUEnview. UUEnview has a big advantage over the standard uuencode in that it can directly mail or post files from the command line, either uuencoded or as proper MIME messages.

Both tools can be installed to replace the standard programs uuencode and uudecode, mimicking their command-line syntax, while adding full power to both the encoding and, in particular, the decoding process.

For users who prefer the mouse over the keyboard, a Tcl/Tk-based interface is also included. Alas, I can usually finish decoding from the command line more quickly than the GUI can start up.



```
OK   message.doc
OK   license.doc
OK   GOLF.JPG
OK   fp-217.zip
OK   number.dat

Save Path: /home/fp
Status:  OK
```

Figure 1. Screen Shot of UUDeview Interface

Since development was done on a Linux box and other Unices, the code is highly portable. In the latest release, the programs are built around an encoding and decoding library. Using this library, a programmer can not only easily write different user interfaces (an example of a trivial decoder is 37 source lines in C), but also can use the decoding power of the library in other applications with a small amount of effort. From the web page (see below), you can currently download patches to the **elm** mail software and the **nn** news reader. After integrating the patches, you can tag the mail and articles as usual, then decode any binaries with the touch of a button.

Once the whole world has switched to MIME, UUDeview can finally “Rest In Peace”, but until then, it can save you some frustration when trying to read binary mail and news articles. UUDeview is distributed free under the terms of the GNU General Public License and can be downloaded from http://www.uni-frankfurt.de/~fp/uudeview/.

Frank Pilhofer is student of computer sciences at the University of Frankfurt, Germany and can be reached at fp@informatik.uni-frankfurt.de. He has finally found a hobby to pull him away from the screen—soaring.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Letter to Bob: Configuring an Intel Linux System

Jon “maddog” Hall

Issue #39, July 1997

Jon “maddog” Hall dreams up the ideal configuration for an Intel Linux system in this fantasy letter to Bob Palmer, President and CEO of Digital Equipment Corporation.

Dear Bob:

I have finished configuring your Intel Linux system for you. I think you will be highly pleased with the resulting capabilities.

First, I split the disk space up into a small amount for Windows for Workgroups and WNT, with the rest of the space devoted to Linux. I did this so that you could do a triple-boot, booting either Windows for Workgroups, WNT or Linux. I know you like to experiment with those other operating systems, and when you have visitors from Redmond you like to at least pretend you run their operating systems from time to time, so having the triple-boot capability is handy. This also gives you access to their new 32-bit applications, as well as ensuring that their old 16-bit applications work without a hiccup.

I continued with a basic installation of Red Hat Linux (Red Hat, Inc., Durham, NC), installing the FVWM-95 window manager, which gives a Windows 95 “look and feel”. This will also help with surprise visits from your Washington friends, but I added the “accelerated X” window server and the Common Desktop Environment (CDE) desktop from Xi Graphics, both of which are available from WorkGroup Solutions of Aurora, Colorado. By using these products you can run those really great 3D modeling programs on your Digital UNIX server, and display them on your Linux box using the optional OpenGL extensions. You also get exactly the same look and feel of your Digital UNIX desktop on your Linux system. I am sure this will make you more comfortable with Linux, since CDE is used in your existing environments.

From time to time I know you like to run breadboard simulation applications that run only on a Mac, just to keep your hand in electronic design. I have installed a Macintosh emulator called “Executor 2” from ARDI (Albuquerque, NM) on the Linux box, and tied that into CDE so you can easily launch it. Likewise for those dull moments, you can execute Wabi (available from Caldera, Inc.) and run solitaire there, as well as a whole bunch of other Windows 3.1 applications which you can launch directly off the W3.1 part of the disk, since Linux can mount MS-DOS file systems. And Bob, you will be amused to find out the “disk copy” function under W3.1 actually works faster with Wabi than it does running native on the hardware. In fact, a lot of the Windows programs running under Wabi seem to execute faster than they do in native mode. Perhaps that is only my perception (I do admit a bias), but it could also be that Wabi takes advantage of the buffer cache of Linux in its input/output operations, as well as the virtual address space and virtual memory protections of Linux.

Speaking of faster, I would not have done all of this work if I could not give you a better environment than you had before, and with the addition of two more tools, I believe I have accomplished that goal. Of course, I realized you still wanted to take advantage of the power and high availability of your 64-bit Digital UNIX system, with its huge address spaces, fail over capabilities and over 5,000 commercial applications available. On the other hand, I know you want to minimize the network traffic over the Internet and create a seamless environment in which you get the most compute power on your desk for the least amount of money. Therefore, I enlisted the aid of Empress Software, Inc. of Markham, Ontario, Canada and Platform Computing of Toronto, Canada. Empress has a distributed a relational database that runs on 1500 Unix platforms, and when it can take advantage of a 64-bit environment (as in the case of Digital UNIX), it does so. This is particularly important (as you know) when you are trying to process large unstructured binary objects. In addition, the Empress RDBMS can attach to an Oracle Parallel Server database and extract information, bringing it directly back to your Linux system. This allows you to intermix the SQL calls with some of the data coming from the Empress database and some coming from the Oracle database, but only the pertinent data coming back directly to your Linux system with the least possible impact on the network. Sure beats grepping those terabyte files over NFS, doesn't it?

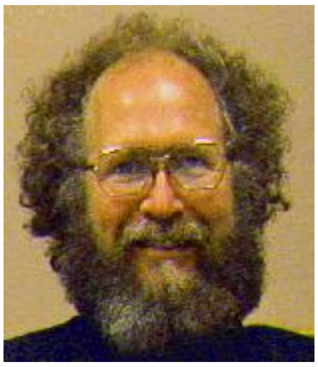
Finally, the use of Platform Computing's Load Sharing Facility (LSF) has me really excited. You know Digital UNIX has clustering capability with really fast recovery from a variety of failures, but Platform Computing has developed software that allows you to have a cluster in a heterogeneous environment. Supported on almost every Unix platform (and even Bill's WNT), it allows you to execute from your Linux system a program on whatever system is best able to run that program, with the output of the application coming right back to your Linux system as if it was executing locally. So, for example, if you were executing GIMP or Emacs, you would probably be executing it on your desktop Linux system, but if you wished to execute PV-WAVE (which runs on both platforms), LSF might transparently run that on the Digital UNIX platform to take advantage of the Alpha's superior floating point capabilities. Or, if you wished to execute that specialized accounting package which only runs on Digital UNIX, by typing in the application's name, LSF realizes it works only on Digital UNIX and finds the least loaded Digital UNIX system (taking into account CPU, I/O, memory constraints, etc.) to run it on, and send the output back to your system. And the accountants will be happy to know LSF can also do that batch processing they have been asking for. LSF truly makes use of the network as an extension of your computer.

So Bob, I hope you enjoy the Linux system I have created for your desktop. Whether you choose CDE or Windows 95 as the “Look and Feel” of Linux, you will still be able to have access to these features, since all of these applications run both on Intel Linux and Digital UNIX, creating that seamless environment you keep demanding.

I will have a little time next week, so I will be happy to duplicate this environment for your Hi-Note Ultra II laptop.

Please say “Hello” to Mrs. Palmer for me, and thank her for the cookies.

Sincerely,Jon “maddog” HallExecutive Director, Linux International80 Amherst St.Amherst, NH 03031-3032 U.S.A.


Jon “maddog” Hall is Senior Leader of Digital UNIX Base Product Marketing, Digital Equipment Corporation.

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

CeBIT '97

Belinda Frasier

Issue #39, July 1997

A report from Germany on the world's largest computer fair.

CeBIT is the world's largest computer fair, bringing together vendors and attendees from many different countries. If you picture landscaped fairgrounds with 27 halls for vendors and even more auxiliary buildings with stores and restaurants and then add 650,000 people to the picture, all visiting the location over seven days, you are starting to get an image of CeBIT '97. CeBIT took place in the Messegelände in Hannover, Germany, March 13th to 19th, 1997.

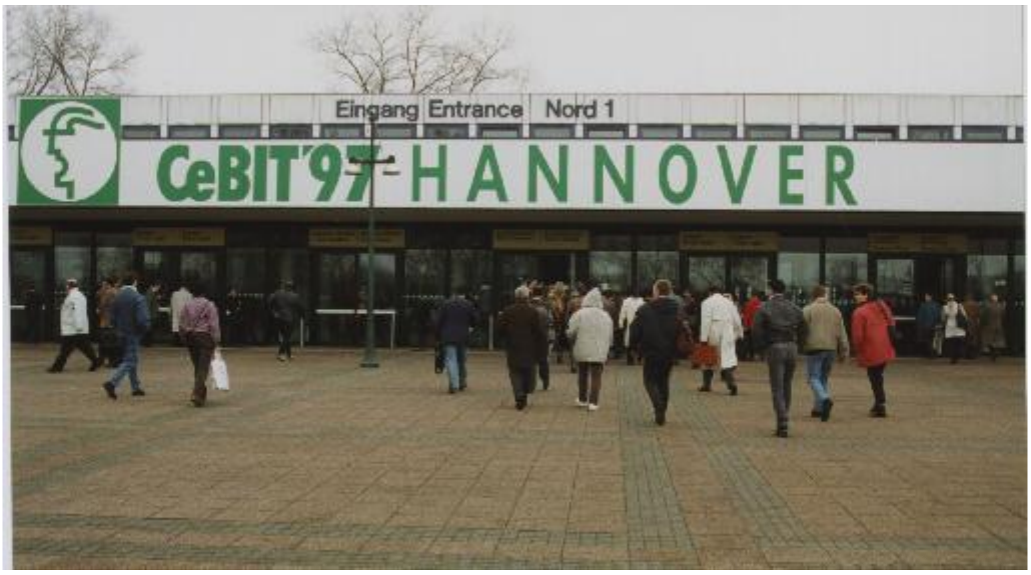


Figure 1. The North entrance to CeBIT

This was my first time attending CeBIT, and my goal was to look at the Linux vendors and possibly talk about Linux to vendors whose software already runs under other Unix platforms. I also wanted to see what such a huge computer fair would be like and contrast it to the US's largest computer fair, Comdex, in Las Vegas, which I have attended the past seven years.

My first stop was Hall 11 to visit Caldera, Inc. Caldera's booth was easily recognizable as a Linux booth because of “Tux” the penguin, (well, a stuffed rendition of Tux) sitting on top of one of the monitors. Caldera's booth was crowded with people every time I visited it.

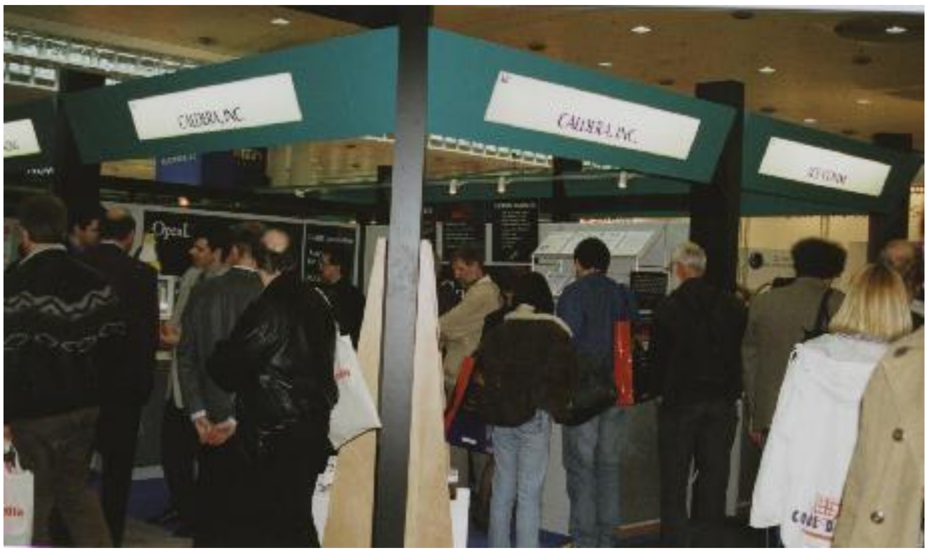
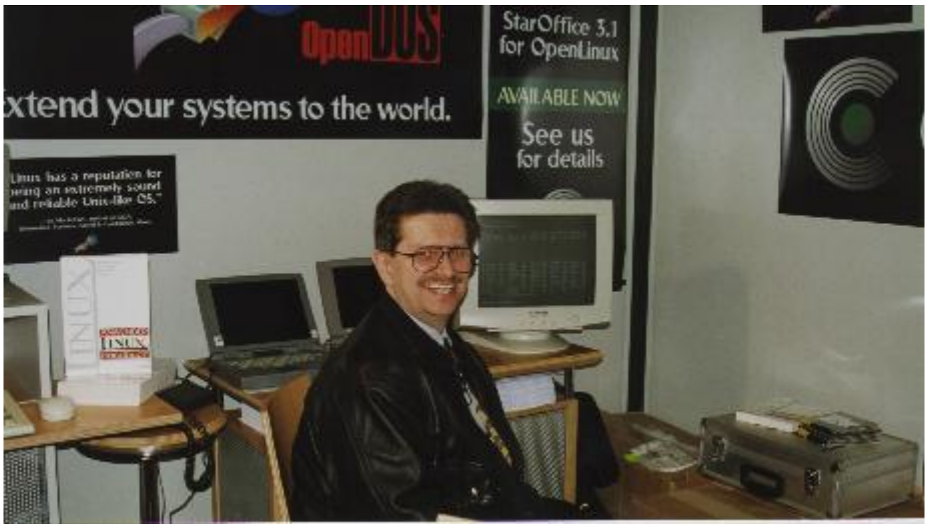


Figure 2. A crowd in Caldera's booth

Attendees were interested in Caldera's OpenLinux products and getting information about Linux and Linux products. The 1500 *Linux Journal Buyer's Guides* given away by Caldera and their affiliated booths during CeBIT, also seemed to be a hit with attendees. Caldera also provided information about OpenDOS 7.01, which is free for non-commercial and educational use. Caldera's booth staff talked about recent announcements such as the upcoming port of Netscape software to OpenLinux, and the port of StarOffice 3.1 to OpenLinux.


OpenDOS
xtend your systems to the world.
StarOffice 3.1 for OpenLinux
AVAILABLE NOW
See us for details
LINUX
Figure 3. Jurgend Plate getting ready for Bayerischer Rundfunk's cameras

A German television station, Bayerischer Rundfunk, filmed a short TV show about Linux at the Caldera booth. The “TV host” Jurgend Plate warmed up for a few minutes while the film crew continued to set up equipment. Before they started filming, after I identified myself as the Associate Publisher of *Linux Journal*, Jurgend hollered to me that *LJ* was “das beste Magazin auf der Welt!” I was told by Sebastian Hetze of LunetIX, that Jurgend Plate had been excited about Linux for years and that his exuberance over Linux was real.

A second Linux stop for me was at the large StarOffice booth in Hall 2 demonstrated on of the company's many different ports, StarOffice on OpenLinux.

At the the third Linux stop, the large Software AG booth, there was a signpost saying Datenbanktechnologie and the second sign down said “ADABAS & LINUX”. Tux sat proudly on top of the workstation here by the Caldera OpenLinux Base. Nathan Guinn gave me a free review copy of the single-user version of LunetIX's ADABAS, an SQL Database, which I passed along to the Editor of *Linux Journal*.

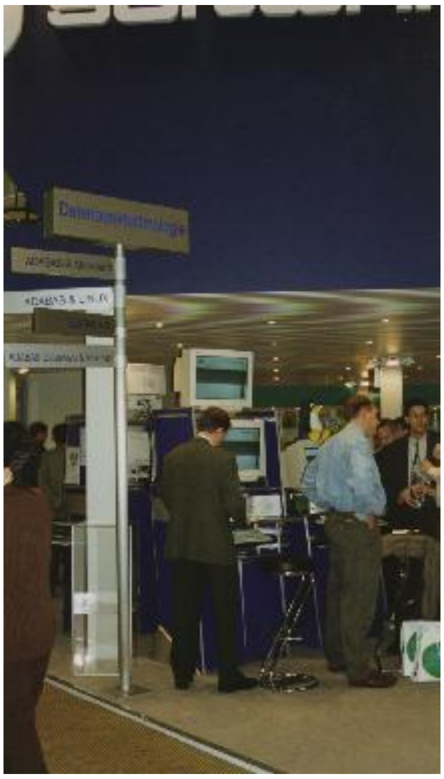


Figure 4: Signpost for ADABAS at Software AG booth with Tux looking over Nathan Guinn's shoulder

Figure 5: Tux at the Caldera and LunetIX display at Software AG's booth

A fourth company with a Linux product was NAG Ltd, which, among its other products, provided information on their Linux FORTRAN 90 Compiler.

Other companies, such as LST Software, GmBH and LunetIX had representatives at the show, mostly working out of Caldera's booth.

There was some press coverage about Linux. In the special CeBIT section of the Newspaper called “COMPUTER & KOMMUNIKATION” there was a full-page article titled “Linux schultert Microsoft-Anwendungen” which covered the capability of Microsoft Applications to run under Linux using Caldera's Wabi software.

All in all, CeBIT was an informative, busy, intensive, show. Next time I should like to try it without crutches resulting from a sprained ankle. I should also mention the color shows and performances in some booths, including a musical story (D2-Musical) with “Princess Digital, the Queen of the World”, the artistic acrobats at VIAG Interkom and many cabaret-style performances, which added fun, colorful and entertaining diversions to CeBIT.

Belinda Frazier is the Associate Publisher of Linux Journal. She can be reached via e-mail at info@linuxjournal.com.

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Microstation 95 for Linux

Bradley Willson

Issue #39, July 1997

Microstation survived the “Can I Break It” test and gets four and a half stars for its performance.

- Product: Microstation 95 for Linux
- Manufacturer: Bentley Systems, Incorporated
- Retail cost: $3995.00
- Reviewer: Bradley J. Willson

My first test for any application is the installation process. If I can install the package without reading the documentation, that is a good thing. If I can use the software within minutes after setting up, I consider that another good quality. Finally, if I can use the application without reading the documentation, I consider it a truly intuitive application.

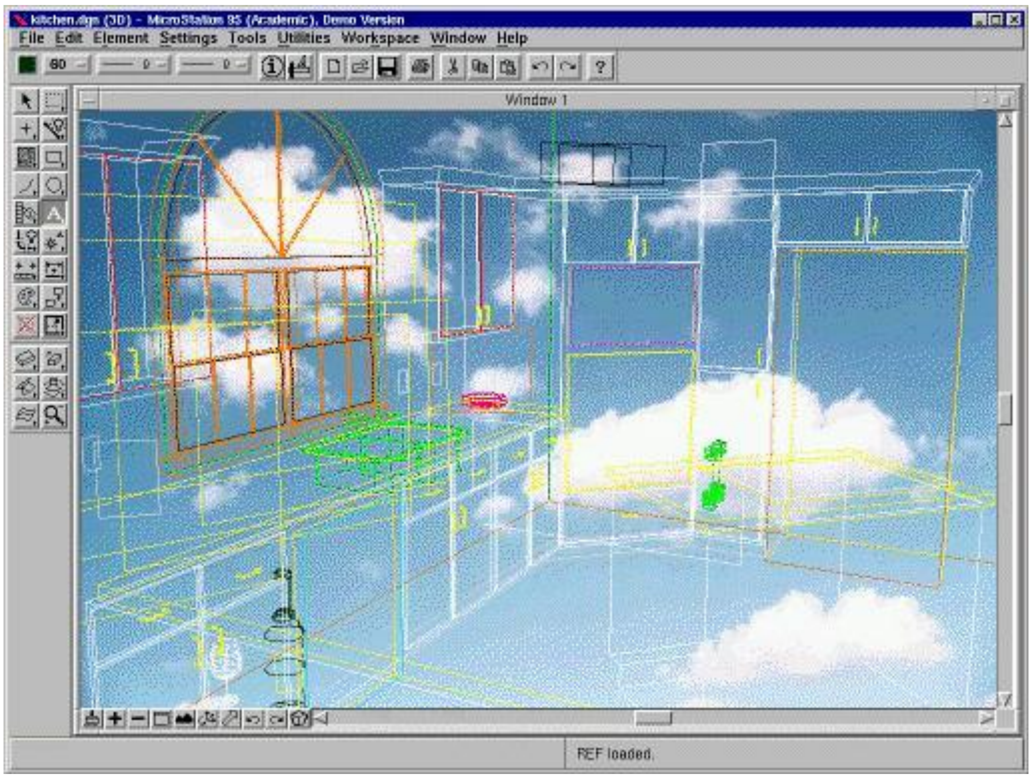
Microstation 95 met or exceeded most of my expectations in each of these three areas.

As a consultant, I get numerous opportunities to fix things, but as a product reviewer, I get to see if I can break something. Microstation survived the “Can I Break It” test and gets four and a half stars for its performance.

Description

Microstation 95 is a commercial-grade CAD application for architectural, engineering, construction, geographic information systems and mechanical design professionals. Microstation 95 provides the 2D and 3D environment tools necessary to design and add detail to a drawing quickly and efficiently.

Furthermore, it provides network connectivity for work group environments and a database server with connectivity to numerous Unix- and DOS-based database applications.


kitchen.dgn (3D) - MicroStation 95 (Academic), Demo Version
File Edit Element Settings Tools Utilities Workspace Window Help
Window 1
REF loaded.

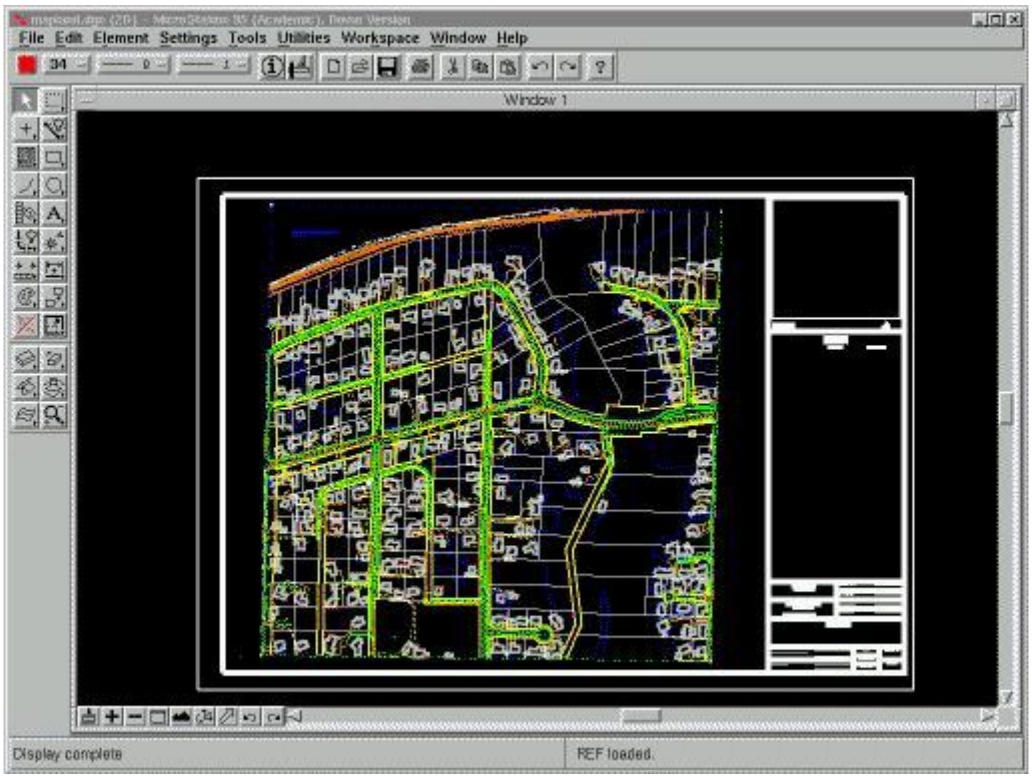
Microstation 95 looks and acts like a typical Microsoft compatible product, with familiar drop-down menus, button pop-up tips and hypertext help. It offers the option of a Windows or Motif-style interface. The Windows option is true to form, even to the extent of using the same names and parameters found in the Windows Control Panel “Colors” option.

I chose the Motif configuration, because it is more flexible than the Windows option. There are no discernible performance or feature penalties for using either interface. Active windows are handled in the same manner as in MS Windows with one notable exception: if one window is too large, it will not arrange the windows to fit the screen, prompting the user to adjust the window size instead.

The toolbox buttons display tool tips, which are helpful because not all of the button graphics are clear as to their function.

Installation

Immediately after receiving the Microstation 95 review kit, I loaded the CD and installed the software. Upon executing **setup.exe**, I encountered an error that had to be resolved before I could continue. Adding a symlink to **libncurses.so.1.9.9e** fixed the problem, and the installation resumed. The installation prompted me with the customary questions: “Where do you want it?” and “How much do you want to install?”


mapsout.dgn (2D) - MicroStation 95 (Academic), Demo Version
File Edit Element Settings Tools Utilities Workspace Window Help
Window 1
Display complete

REF loaded.

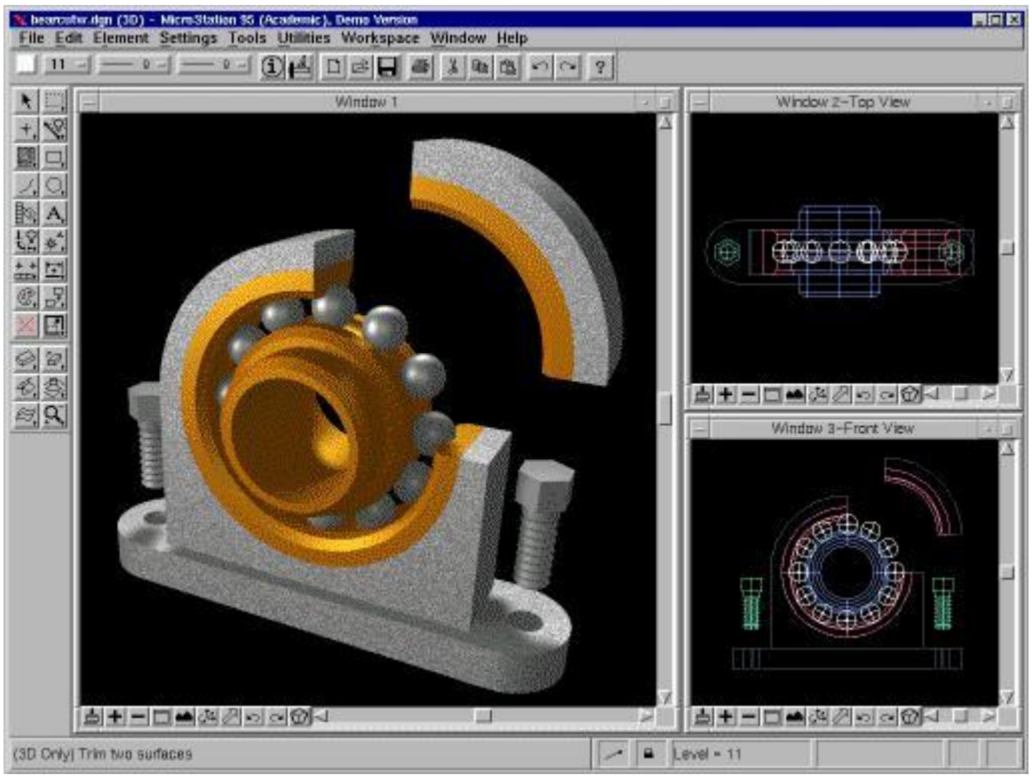
The user preferences and system configuration are taken care of from within the running application. Overall, the installation is fast, easy and straightforward. The process takes about 30 minutes to complete and the package uses a miserly 50MB of the hard drive.

Tech Support

The only documentation provided with the evaluation copy is in the form of the help files and a Microsoft Windows-based sampler CD. This did not present any major problems, because the interface is generally easy to understand. I did call the company's toll-free number to inquire about tech support, only to find that a call to the tech support people would cost me a long-distance call—not what I wanted to hear.

Comparison with Other Products

To compare an application like Microstation 95 to mainframe CATIA may seem unfair at first, but in reality, I find Microstation 95 to be more flexible and more agile in its execution. In the short time I had to review Microstation 95, I was impressed with its capabilities, speed and efficiency in comparison to its mainframe cousin.


bearcutw.dgn (3D) - MicroStation 95 (Academic), Demo Version
File Edit Element Settings Tools Utilities Workspace Window Help
Window 1
Window 2-Top View
Window 3-Front View
(3D Only) Trim two surfaces
Level = 11

I chose to use the tutorials to both learn the system and to judge its suitability for the task. I found the examples to be very informative and a great way to push the test envelope. I learned quickly to turn off the automatic save on exit feature, so I could return to the example later and try new torture methods on it.

The true test came when I began rendering the examples. At each level of display, the load requirements increased and the display performance declined. What surprised me was Microstation 95's speed. I am accustomed to waiting several minutes for a simple shading operation in CATIA. I was not expecting to see a complex model rendered in less time, but that's what happened..

The AccuDraw and SmartLine features enhance productivity by producing intelligent elements that are easily placed and modified. Bi-directional associative dimensioning makes detailing fast and easy. CATIA offers similar functions, but Microstation makes them more readily available to the user.

Company Profile

Bentley Systems, Inc. was founded in 1984 by Keith and Barry Bentley. It is a privately held corporation with over 400 employees headquartered in Exton Pennsylvania, Hoofddorp, The Netherlands and South Melbourne, Australia. They maintain a web site at http://www.bentley.com/. Phil Chouinard, Product Manager, Foundation Products, can be reached by phone at 610-458-5000 or e-mail at phil.chouinard@bentley.com.

Installed Base

The review kit states that there are over 200,000 users of Microstation 95 in over 15,000 companies and organizations around the world. They include Boeing, AT&T, American Airlines, Westinghouse Electric Corporation, U.S. Department of Commerce, Union Carbide C & P Company plus numerous other commercial users and government agencies.

Platforms

The review was performed on an AMD 5x86-P75 133MHz machine with 20MB RAM, running Linux 2.0.0. Display was handled by a Number 9 Vision 330 1MB video card.

Bentley Systems, Inc. also offers Microstation 95 for DOS, Windows 3.1x, Windows 95, Windows NT, OS/2 Warp and PowerPC or Dec Alpha AXP PCs running Windows NT. Microstation 95 will run on 80386 (with a math coprocessor), 80486, Pentium, DEC Alpha and PowerPC processors, with a minimum of 8MB RAM for the DOS package and 24MB minimum RAM for DEC Alpha or PowerPC.

More Figures

Bradley J. Willson (bc115@scn.org) currently designs and troubleshoots tooling for the Boeing 777 program and fills the chair of chief cook and bottle washer for Willson Consulting Services. His friends understand and forgive his addiction to computer technology, while others wonder how he can stand the countless hours he spends staring at screens. According to Bradley, the secret is attitude—and maybe a mild case of radiation sickness.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Learning the bash Shell

Danny Yee

Issue #39, July 1997

It begins with the basics, not assuming the user has any experience with other shells or even with Unix, and would make a decent introduction to shell programming.

- Author: Cameron Newham & Bill Rosenblatt
- Publisher: O'Reilly & Associates
- Price: US$27.95
- ISBN: 1-56592-147-X
- Reviewer: Danny Yee

Like the other books in O'Reilly's *Learning* series, *Learning the bash Shell* is aimed at novices but is also useful to experienced users. It begins with the basics, not assuming the user has any experience with other shells or even with Unix, and would make a decent introduction to shell programming. The coverage is, however, comprehensive, with later chapters covering the more obscure features of bash and the more complex aspects of shell programming.

Many of my friends think the idea of a whole book devoted to a shell is bizarre, but that is more of an objection to bash (which **is** a bit on the baroque side) than anything else, and I won't enter into the relative merits of different shells. If you are, say, a novice Linux user, then *Learning the bash Shell* contains more than you are likely to want or need at first: the brief summary in a more general book such as *Running Linux* will be enough to get you by. If you start to do any sort of serious shell programming, however, *Learning the bash Shell* would be a most useful volume. Whether those who are familiar with other shells find it worthwhile will largely depend on how comfortable they are with the bash manual entry.

Danny Yee received review a copy of the book mentioned from O'Reilly & Associates, but has no stake, financial or otherwise, in their success. He can be reached at danny@cs.su.oz.au.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Source Code Secrets

Phil Hughes

Issue #39, July 1997

If you are a savvy OS person, you've already figured that out from the names of the authors; Bill and Lynne Jolitz are the creators of 386BSD.

- Authors: William Jolitz and Lynne Jolitz
- Publisher: Peer-to-Peer Communications
- Pages: 530 (hard cover)
- Price: $49.95
- ISBN: 1-573988-026-9
- Reviewer: Phil Hughes

Before you get too excited about a new book about the Linux kernel, let me clear the air—*The Basic Kernel* is about 386BSD. If you are a savvy OS person, you've already figured that out from the names of the authors; Bill and Lynne Jolitz are the creators of 386BSD. So, why review it? Because it is a good book that covers the internals of operating systems design—and 386BSD, the system it covers, is available in source code.

The Basic Kernel is the first volume in a series entitled *Operating System Source Code Secrets*. The second book in the series is titled *Virtual Memory*. The first thirty pages of the book explain why such a book is necessary, and define both what a kernel is and the notation used in the book.

The book then talks about what code needs to be done for the kernel, presenting code fragments with explanations. About 60 pages cover the assembly code necessary to interface to the 386 processor. Not being a 386 assembly language programmer (but having extensive systems programming experience), I had some problems here. The explanations were clearly for someone with an assembly language background. I would have liked to see more in the way of “how it works” explanations. However, if you are interested in 386 code, it is probably fine.

I felt more comfortable with the next chapter, covering C code. The beginning of that chapter is typical of what follows, so I will use it as an example. The trap.c routine is introduced, along with a listing and brief explanation of its functions, followed by a detailed explanation of what the functions are, how to call them and how they are implemented. The implementation details consist of code fragments with text explanations.

The book continues in this style, covering i386/trap.c, i386/cpu.c, kern/config.c, kern/malloc.c, kern/fork.c, kern/exit.c, kern/sig.c, kern/cred.c, kern/priv.c, kern/synch.c, kern/lock.c, kern/execve.c and kern/descrip.c. If you have looked at or worked with a POSIX kernel, you can probably guess what functions are found in these files.

Three appendices cover kernel source organization, network-level security and dynamic make files. The book ends with answers to the exercises included in the chapters, and an index with entries for all the functions described.

The book is well-written, and, I expect, accurate. If you want to know how the 386BSD kernel works, this book will tell you. And, if you want to know how to implement low-level functionality that requires dealing with 386 hardware, this book has a wealth of information on the subject.

What this book won't tell you is how Linux is implemented. While the same sort of tasks have to be done in Linux, different design decisions were made for 386BSD than for Linux. As I was reading the text, many times I was thinking, “Gee, I wonder if this is done the same way in Linux?” If that comparison is of interest to you, buying both *The Basic Kernel* and *Linux Kernel Internals* (reviewed in *Linux Journal*, December, 1996) would probably fill your needs.

Phil Hughes is the publisher of *Linux Journal*.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Creating a Multiple Choice quiz System, Part 3

Reuven M. Lerner

Issue #39, July 1997

We continue to improve our quiz file program, by adding an editor.

Last month, we continued looking at ways to improve the user interface of our CGI quiz engine. This engine is based on the **QuizQuestions.pm** Perl module and allows us to create a number of different multiple-choice quizzes. Each quiz is stored in a separate ASCII text file on the server's file system, and thanks to the abstraction layer provided by the **QuizQuestions** object, we are able to ignore the way in which the information is stored and focus on the quiz itself.

Or can we? As we saw last month, it is relatively easy for us to create a quiz using HTML forms and a CGI program. Indeed, we created such a program without much trouble; still, as I pointed out at the end of last month's column, our job is only half done. Using an HTML form and CGI program simplifies the process of creating quiz files and reduces the potential for error. However, of someone wants to modify a quiz that he has already created, he still has to understand our file format and make sure not to disturb it. It would be nice to have a single program which allowed users to create and modify their quizzes, saving them from having to work directly with quiz files.

Such a program would be desirable for its error-free, user-friendly quiz creation. We use text files to store information with tabs or other special characters separating fields within a given record. Fewer problems will ensue if we can provide a graphical interface that restricts the type of data that our users can input. In addition, a quiz editor would empower the designers and producers on our site—who, at an increasing number of sites, are kept separate from the technical staff responsible for creating CGI programs and keeping the network up and running. While you might be an experienced system administrator who knows the difference between tabs and spaces and doesn't flinch before editing /etc/fstab or a Makefile, most content-oriented people on a site are not experienced users. If we can provide a tool that creates and modifies quizzes without the need to learn Emacs, come to us with questions or tear their hair out in frustration, then why not?

Last month, we looked at a simple program, **create-quizfile.pl**, which took information from an HTML form and correctly formed it into a quiz file. This month, we take the basic idea behind that quiz creator and look at how we can write a quiz editor. That is, our new program will allow us to create quizzes, just as the previous one did, but will also let us edit quizzes by modifying the text of questions and answers or deleting existing questions from the quiz file.

From the above description it seems as though our basic unit of operation is a line of text in the quiz file that will contain a question. The easiest way to handle these data is to keep track of them in an array of strings, which we call **@lines**. This means **$lines[0]** is the first non-comment, non-whitespace line in the quiz file—in other words, the first set of questions and answers. For example, to find the third answer to the fifth question, we call up **$lines[4]**, split it across tabs using Perl's “split” operator, and read the fourth element of that array. Each line of the quiz file is of the format *question answer1 answer2 answer3 answer4 correctAnswer*, with each field separated from its neighbor by a tab character. Each line of the quiz file can be represented by a string containing the tabs or by a list created by splitting the string across the tab character.

Luckily, we don't have to think much about the format of the quiz file, thanks to the **QuizQuestions** object module we have used so far. When we create a new instance of **QuizQuestions**, we effectively create a new, blank quiz. To add questions to that quiz, we use the **addQuestion** method, which expects to receive six arguments—not surprisingly, the same arguments as appear on each line of the quiz file.

To create a brand-new quiz our program, called **edit-quiz.pl**, must create an HTML form containing elements into which the user can enter one or more questions and answers. To edit an old quiz file, **edit-quizfile.pl** has to create an instance of **QuizFile** corresponding to a quiz file already existing on disk. It then must read the questions from that quiz file, turning each question into a set of HTML form elements. This lets the user edit the questions and answers, delete existing questions and add new ones.

But wait a second—if **edit-quizfile.pl** is going to create the HTML form that the user will use for editing a quiz, which program will take the contents of this form and actually do something with it? After all, CGI programs are single operations: they receive input, perform some processing and produce output. It seems if our program produces an HTML form as output, it cannot also accept input from that form and save it to disk. The secret here is that **edit-quizfile.pl** can accept its own input by expecting to be invoked twice. On the first invocation, it creates the form to be used to edit quizzes, and on the second invocation, it saves the submitted data in the form of a quiz. Our program thus performs two different actions on two distinct occasions.

Why not simply write two separate programs, one to display the form and a second to process it? We certainly could have handled it in this way, but this would restrict us to a single iteration of editing. By keeping all of the quiz-editing features within a single program we can create an editing loop that allows us more than a one-shot deal. We can reuse the code for displaying the current state of the quiz file, because we always display the quiz's file state when our program is invoked. By keeping all the code within one program and simply putting the display code at the bottom of this program, everything becomes a bit easier to understand and maintain.

Displaying the Quiz

Before we get to work coding, let's figure out how all of this works. We invoke **edit-quiz.pl** in a number of different ways, using both **GET** (i.e., entering its URL in a browser window or clicking on a hyperlink within a page of HTML) and **POST** (i.e., when we click on a **submit** button at the bottom of an HTML form).

If **edit-quiz.pl** is invoked using **GET** without any arguments, we are asked which quiz we wish to create or edit. Entering the name of the quiz file and pressing Return submits it to the program, which then receives this argument—again using **GET**--in the query string.

The query string, for those of you relatively new to CGI programming, is the term for anything following the question mark in a URL. It allows the passing of simple arguments to a CGI program without having to use **POST**, a more complicated and sophisticated protocol for passing information. Thus, if someone invokes the program:

```
http://www.fictional.edu/cgi-bin/program.pl
```

the query string is null, because there is no argument. But if someone invokes the CGI program:

```
http://www.fictional.edu/cgi-bin/program.pl?foobar
```

the argument is **foobar**. If we are using **CGI.pm**, a Perl module for writing CGI programs (available from CPAN at http://www.perl.com/CPAN), we can theoretically retrieve the contents of the query string using the **query_string** method, as in:

```
my $query = new CGI;
my $query_string = $query->query_string;
```

For reasons that I don't completely understand, the **query_string** method returns the query string with a prepended **keywords=**, as if the query string had been submitted to our CGI program in an HTML form element named **keywords**. While this can sometimes come in handy, for the most part, I find it a surprising quirk in an otherwise excellent package.

If **CGI.pm** is going to treat the query string as a parameter named **keywords**, we have to retrieve its value in the same way as we would other parameters, namely:

```
   my $query = new CGI;
   my $query_string = $query->param("keywords");
```

which might seem a bit odd at first, but you get used to it.

We determine whether our program was invoked via **GET** or **POST** using the method **request_method** within **CGI.pm**. In other words, we can do the following:

```
   my $query = new CGI;
   my $request_method = $query-7gt;request_method;
```

At this point, the variable **$request_method** contains either the string **GET** or **POST**, depending on how the program was invoked. The main difference between these two invocation methods is how arguments are passed to the program: **GET** sends all of the variable values in the query string, while **POST** sends them via stdin, the file handle associated with standard input. Luckily, CGI.pm frees us from having to deal with these methods and invisibly hands us the parameters regardless of their source.

In any event, if our program is invoked with any value in the query string, we print out an HTML form containing the contents of the quiz file with that name or a blank HTML form allowing the user to create a quiz of that name. You can see that program in Listing 1.

There are several things to note in this program. First of all, we need to tell the program the maximum number of questions that each quiz can contain. We do this by setting a global variable, **$MAX_QUESTIONS**, at the top of the program. Either very short or very long quizzes can be allowed for by changing this variable.

Also, notice how we manage to create a page of HTML that invokes our program with an argument in the query string. We use the <ISINDEX> tag, which has been all but forgotten on the Web, mostly because it creates an ugly text box whose instructions are difficult or impossible to change and rarely relevant to the subject at hand. Nevertheless, <ISINDEX> comes in handy if you want to provide a program with a mechanism to feed a user-defined argument to itself.

In addition, we create a new instance of **QuizQuestions** based on the name of the quiz that we received from the user in the query string. Once the instance of **QuizQuestions** is created, we instruct it to load its contents from disk. Of course, if this is a new quiz, then there is nothing to load, and this is noted in the error message returned by the **loadFile** method. We don't care if there was an error opening the file—if the quiz file exists, the contents are displayed in the HTML form, but if it does not exist, it is treated as a new quiz.

Of course, it is not a good idea to ignore error messages altogether. But the error messages returned by the **loadFile** method are fairly primitive, indicating whether the file was successfully loaded. Better error messages might distinguish between an inability to find the file in question, a quiz file that exists but cannot be read and a quiz file containing errors. But for now, this is all we've got, so we will have to live with it.

We insert the current value of each HTML form element by placing the value inside a variable. One of the nice things about Perl is that uninitialized variables default to the empty string (**""**). This means that if we have not set a particular question or answer, things don't crash. Rather, since we get the empty string back from the variable, we can stick the variable into the form element's **value** attribute, thus resetting the form element's value.

We use a bit of cleverness to indicate which element of the selection list (which is used to indicate the correct answer) should be selected by default. Here is the code:

```
my $letter = "";
foreach $letter
("a","b","c","d")
{
   print "<option ";
   print "selected " if ($letter eq $correct);
   print "$letter>$letter\n";
}
print "</select>\n";
```

In this code, we simply iterate through all four possible correct answers, inserting the word **selected** inside of the **<option>** tag where it is appropriate.

As you can see, it is not difficult to create a program that displays the contents of a quiz file. If we want to create an editor, we need to write the second half of the program, namely the part that takes the submitted form contents and saves them to disk. Luckily, the way we have organized our HTML form makes this fairly trouble-free.

Editing the Quiz

The part of edit-quiz.pl that handles saving information is invoked via **POST**, when the user clicks on the **submit** button at the bottom of the page. At this point, the HTML form elements are sent to **edit-quiz.pl** and made available using the **param** method from CGI.pm.

So when edit-quiz.pl is invoked using **POST**, we simply need to do the following:

1. create a new instance of **QuizQuestions**
2. iterate through all of the HTML form elements submitted
3. turn these form elements into new questions
4. save **QuizQuestions**

The simplest way to do this is to loop through all of the possible elements. We know how many elements might exist, thanks to the **$MAX_QUESTIONS** global variable. Thus, we can do something like:

```
my $counter = 0;
foreach $counter (0 .. $MAX_QUESTIONS)
{
   # Add question number $counter
}
```

Adding a new question is now accomplished by using the **addQuestion** method from within **QuizQuestions**. Once we create an instance of **QuizQuestions**, we can add a new question by invoking the method, and passing it the question text, the possible answers, and the correct answer as arguments. Given the names of our HTML form elements are regular, we can expand the above loop as:

```
# Create an instance of QuizQuestions
my $questions = new QuizQuestions($quizname);
# Add questions to $questions
my $counter = 0;
foreach $counter (1 .. $MAX_QUESTIONS)
{
   # Only handle as many questions as were filled
   # in, by
   # checking to see if the question was entered
   last unless ($query->param("question-$counter")
         ne "$counter");
   # Set the question
   my @question =
                 ($query->param("question-$counter"),
                 $query->param("answer-a-$counter"),
                 $query->param("answer-b-$counter"),
                 $query->param("answer-c-$counter"),
                 $query->param("answer-d-$counter"),
                 $query->param("correct-$counter"));
   # Add the question to the quiz
   $questions->addQuestion(@question);
}
```

The above loop should look familiar if you looked through last month's column. That's because the loop is lifted from **create-quiz.pl**, I have changed a variable name and modified the condition on the “last” statement. This ensures that the loop will exit if the text of the quiz question is the same as the quiz number, since each quiz question is set to its number in our editor.

Now that you have seen how the heart of edit-quiz.pl works, take a look at the entire program in Listing 2. As you can see, it did not take many modifications to make this program work. That's because the bulk of the work is done by objects already created. CGI.pm takes care of reading the HTML form elements and handing them to us in a nicely packaged format, while QuizQuestions.pm takes care of loading and saving the questions from the quiz file. Now we don't have to get our hands dirty. (And neither do our users, who can now create quizzes without having to worry about formatting issues.)

I have also done a bit of reshuffling with the original segments of **edit-quiz.pl**, such that we always see the results of our editing work and can make additional changes to the quiz. This is somewhat better than the standard Web interface, in which saving also means quitting. Here, there isn't any real **exit** button, since saving brings you back to where you were.

There are a few minor problems with the current versions of these programs. In particular, it is potentially dangerous for a user to enter a pair of quotation marks (**""**) inside one of the text fields. A browser might have difficulties determining which quotation marks belong to the string and which set the string's boundaries. (However, Netscape 3.0 on my Linux box appears to handle this just fine, much to my surprise.)

Although we forget it most of the time, we continue to be haunted by our old friend the tab character, which still separates the fields in our quiz file. If a user were to enter a tab into one of the text fields in our quiz editor, the quiz file's format would be damaged and would cause problems. A solution to this would involve iterating through each of the HTML form elements handed to us by CGI.pm and removing any tabs. Better yet, we could replace them with spaces.

Notes

Next Month

Next month, we will look at hybrid templates that you can create with HTML and Perl, thanks to the magic of the Text::Template module. By embedding programs inside of HTML files, we can include both static HTML and programming elements. This can be a great advantage when you want to produce output using a program, but don't want to lock the HTML styling inside of a program, since your site's editors and designers will eventually want to change the content and style of the program's output.

Reuven M. Lerner is an Internet and Web consultant living in Haifa, Israel, who has been using the Web since early 1993. In his spare time, he cooks, reads and volunteers with educational projects in his community. You can reach him at reuven@netvision.net.il.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Letters to the Editor

Various

Issue #39, July 1997

Readers sound off.

Wrong School, Right Side of the Mountains

I just received several copies of the April issue. On page 62 I see an article written by me with the subtitle: “Assembly language is a wonderful tool for teaching about how computers work. Professor Sevenich explains how it is used at WSU.” I am at EWU, *not* WSU. —Richard Sevenich from somewhere east of the Cascades rsevenic@blinn.sirti.org

XFree86 with the Matrox Millenium Board

Your article on Lectra Systèmès CAD/CAM systems (Issue #36, April 1997, page 53) is very interesting. I understand you are having problems with the current version of XFree86 with the Matrox Millenium Board. Here's a possible solution:

The XFree86 Organization has recently released the new 3.2 version of XFree86, which is also known as X11R6.1. It now supports about 320 different cards, including the Matrox Millenium, and several Chips and Technologies chip sets commonly used in notebook computers. The site to download the new version of the XFree86 is: ftp://sunsite.doc.ic.ac.uk/packages/XFree86/3.2/binaries/Linux/ix86-Elf/

The web site for further information is: http://www.xfree86.org/. —T. Parker tparker@acy.digex.net

Linuxers Unite!

I am a college student attending Stephen F. Austin State University. I work in a Geographic Information Systems Laboratory (GIS), and we have been using just AIX machines. However, we do have a full-blown Linux PC, and it is great. We were considering upgrading to all Linux PCs in our lab, because they were cheaper and faster than the AIX boxes, but we ran into a problem—the software we need to run to make our GIS maps is not supported by ESRI. We gave them a call, and they told us: “Linux will not be a supported platform. Product ports are user-driven and there are not enough users wanting this OS”. How could this could be true when all you have to do is get on the web to see that millions of people are using Linux? So I need Linux users to e-mail ESRI at buspartners@esri.com and tell them you use Linux and there are many more people using Linux too. ESRI needs to get its head out of Microsoft's world and see what is going on in the real world. —Tred Riggs tred@oak.sfasu.edu

Uplifting News

I was very interested in Eric Raymond's article “Building the Perfect Box: How to Design Your Linux Workstation” (Issue # 36, April 1997, page 16), and it was a good read. But... Under the section entitled “Some pitfalls to avoid”, Eric says don't buy PnP cards—Linux doesn't support them. I would suggest this is rather a bleak way to look at it and that the situation is not that bad. Check out isapnptools on http://www.roestock.demon.co.uk/isapnptools/ for some software I wrote to configure PnP devices. This brings PnP hardware to the same state as configuring jumpers, i.e., you still need a driver.

Currently, I have a PnP internal modem and network card running quite happily using isapnptools to configure them on boot.

Debian appears to have picked this up as well. —Peter isapnp@roestock.demon.co.uk

Distributions Question

I read the descriptions of the various distributions of Linux in your 1997 Buyer's Guide. I currently use Slackware and was considering a new distribution, possibly Debian. In the review starting on page 128, Phil Hughes states “... as far as I know, no commercial software packages are available in the Debian format.” I am confused. What difference does the format really make? If company X supports kernel Y and the distribution supports kernel Y, shouldn't the product of company X run on that distribution?

I understand that the installation may not be as pretty as if it were written for a package system supported for the distribution. The software, however, should work with the kernel, i.e., if WordPerfect exists for Linux kernel ver. 2.0.0 on, say, Red Hat, should not WordPerfect work for any distribution using kernel version 2.0.0 or am I missing something? —Thomas L. Gossard tgossard@ix.netcom.com

There are two considerations: loading the package and licensing. If the package comes in a format easily unpacked on any system (such as the gzipped tar format used by Slackware), unpacking is not a problem. You may, however, need to deal with versions and locations of libraries and other files to get it running.

If the software is packaged in Red Hat's .rpm format, you need a utility to unpack it. Debian, for example, includes such a package. Here at LJ, we are in the process of converting our systems from Slackware to Debian. As part of this conversion, we are adding office suite software for some of our users. We installed a copy of Applixware on a Debian system with no problems. (And Erik Troan of Red Hat informs us that Applixware will run on other distributions as well.)

You mentioned WordPerfect, which leads us to the licensing issue. Caldera has a licensing agreement with Corel Corp., makers of WordPerfect, which states that you can run the Linux version of WordPerfect only under Caldera's flavor of Linux. My understanding is that the other applications Caldera has facilitated porting to Linux are not licensed this way—you are free to run them with any flavor of Linux. —Phil Hughes info@linuxjournal.com

Pure Java

I read your article about The Linux Appliance (“From the Publisher”, Issue #36, April 1997, page 12) with interest. Right offhand, I can't help but ask myself:

How can such an appliance compete with a “pure Java” system as an appliance? What kind of marketing technique might be used to offset the power of Microsoft? How might we get some major hardware makers (like Sony, Panasonic, etc.) interested?

Is there anything I can do, or anyone I might contact, to help this along (purely on a part-time basis; I have a full-time job as a mainframe/NT system support analyst; if it became a money-maker, I could quit, of course)?

I don't have any business or manufacturing experience, but I have wanted to contribute something good to Linux for a long time. This might be a possibility. —Chuck

I'm not the person ready to manufacture and market this appliance so I am hoping someone else will seriously address these points, but here is my input.

A pure Java system might be a winner, but it could also be a dead end. An addition to Java or a move to something other than Java would result in an obsolete system. On the other hand, a programmable computer makes it easy to follow software changes. In addition, upgrading from an Internet Appliance to a useful computer for word processing and such is also possible.

As for the power of Microsoft, the sales approach is what could make the difference. Anyone can buy a PC/Microsoft system from a catalog or retail store but for many, this is a scary proposition. While computers have come a long way toward being easy to set up and use, offering the necessary support with a system like this could make a big difference. Much like getting a price break on a cellular phone when you purchase air time, ISPs could offer a price break on these systems with connectivity contracts. The ISP would then be there to support it.

As for the possibility of Sony or Panasonic getting interested, if you know the right people to talk to, point them at me. The idea of Linux on the Sony Playstation has already been suggested. —Phil Hughes info@linuxjournal.com

Applixware Review

I appreciated your review of Applixware in the April 1997 issue of the *Linux Journal*. I do feel obligated to point one large error—Applix data is NOT provided with the 4.3 release, according to Lisa Sullivan at Red Hat.

There are some people (Cameron Newham, cameron.newham@nomura.co.uk) working on interfaces to databases that do exist on Linux, but the full-fledged Applix Data interface is not likely to exist in the near future, a victim of lack of support from the big-3 database vendors—at least, that is the reason in my opinion.

Applixware has a chance to do either a lot of good or a lot of harm to the Linux community. It is not quite ready to replace MS Office. Filters and data access, as well as performance with embedded graphics, are areas that need work.

If people expect it to do everything, try it, and are disappointed, they may end up dismissing Linux as junk—a tragic mistake. —Cary O'Brien cobrien@access.digex.net

Many Thanks

Dear Terry Dawson,

I just wanted to thank you for your article “A 10-Minute Guide for Using PPP to Connect Linux to the Internet” that appeared in Issue 36 of *Linux Journal*.

I bought Slackware 2.3 (kernel version 1.2.8) in September of 1995, loaded it on my Micron Pentium-90 and have been experimenting with it ever since. Being more experienced with DOS/Windows, I wasn't successful in getting my PPP connection to work from Linux. I read Matt Welsh's *Running Linux* and Patrick Volkerding's *Linux Configuration and Installation* as well as the PPP HOWTO but had no success until your article. Since then, I've gotten ftp, rlogin, telnet, lynx 2.3 and Netscape Navigator 1.1 to run. Next, it's e-mail (I'm sending you this from my DOS/Windows partition). —Steve Tjensvoldstjen svo@execpc.com

Etherminals

I read your article in this month's *LJ* with much interest. In the March 10th issue of *Computerworld* magazine, Mr. Charles Babcock wrote about building his own network computer from scratch. From the sound of his article, Mr. Babcock had no previous experience with Linux, yet that was the OS he chose for his NC.

Mr. Babcock was not able to keep the price of his NC below $500. However, it is not clear whether he used many resources towards getting the best prices available. Clearly, anyone venturing into this on a large scale would be able to better his price—$1200.

What interests me most about both articles is the fact that such a system already exists—and has existed for over a year. IGEL, LLC has a line of systems called Etherminals—they were even advertised in *LJ* for a while. (That's where I learned of them)

I left a full-time position as a Unix Systems Engineer in August of '95 to help my wife with the acquisition of the veterinary hospital where she had worked (as a veterinarian). One of my first responsibilities was to install a computer network.

That's when I discovered the Etherminals. The model 3x was IGEL's current offering—386sx-40 w/ 8MB RAM, on-board Ethernet (all 3 media types) and the standard compliment of 2 serial and 1 parallel ports. What made these machines so attractive to me was that they booted Linux and X Windows from ROM/NVRAM. These units had no drives—not a floppy, not a hard drive, nothing.

I went out on a limb and bought three of them. Once they arrived, I connected them to the Ethernet I'd run (with the help of a couple of good friends) the week before. In a little over an hour I had everything working like a charm. I think it helped more than a little that the server was running Linux as well. (This was something that I insisted on—it even became a point of contention with two potential vendors of veterinary office applications.)

In the end, we chose a vendor whose application was (and still is) DOS-based. DOSEMU now runs multiple sessions on the server and uses X to redirect the I/O to the Etherminals. It's an acceptable solution—DOS being the weakest link.

The most important point of this whole message is that this solution was arrived at over a year ago. No need to wait for SUN or ORACLE. It's here today. —Chuck Stickleman stick@richnet.net

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Advanced search

From the Publisher

Phil Hughes

Issue #39, July 1997

To make this process even more fun, there have been a lot of computer-related problems—all related to our Linux systems.

The end of March is quickly approaching as I write. Here at SSC, March is always an exciting time as it is the end of our accounting year. For the last two weeks Gena Shurtleff and I (with help from others) have been working on a budget for the next year of *LJ*--something that has made some of us very grouchy. [mainly, our publisher --Ed.]

To make this process even more fun, there have been a lot of computer-related problems—all related to our Linux systems. [Note: if you are humor-impaired you may want to skip a lot of this.] First, our main server failed. Linux apparently caused a pin to break off the cable to the external SCSI disk drive. Then, a week or so later, Linux broke a head on the disk drive in our firewall. Next, the fan in our editor's computer started growling at her. On top of all that, various systems in the office were mysteriously crashing or exhibiting very strange behavior in general. The end result was too many hours of downtime, lost e-mail and an unhappy working relationship with our computers.

We began to wonder if Linux was good enough for us, and if perhaps Windows NT might support “automatic pin re-soldering” and “disk-head replacement”.

Once things calmed down we took a closer look at the problems. By the way, *we* refers specifically to Jay Painter, our new systems administrator, Peter Struijk and me. First, we concluded that it probably wasn't the fault of Linux that hardware was breaking. As scary as it is that multitasking operating systems write to disks whenever they think it's a good time, this really isn't a reason for a cable to break.

To address the issue of being down for longer periods of time than we thought appropriate, let's look at some specific cases.

Note: at this time, our various systems ran a host of different software versions (kernels and C libraries), and we were in the midst of converting them to Debian in order to have consistency across all machines.

The Server Failure

The failure of the cable caused the data on the server disks to be scrambled. Fortunately, we had back-up copies of the files, so it seemed like a good time to do an upgrade. We made the logical decision to reload a standard Slackware system (rather than try to change to Debian), restore the user files and be on our way. It turned out to be not so easy. The new load of Slackware had more differences from the old version than we expected. Libraries were different. NFS and NIS were different. Adobe fonts we use for doing reference cards had to be reloaded. Configuration files for **groff** had to be updated. A lot of work was done to get the new configuration talking to all the old systems and to get everything tuned.

Firewall Failure

Possibly inspired by the extra work the firewall was doing during the time the server was down, a head died on the disk in the firewall the next weekend resulting in the loss of a lot of mail. Why was it lost? Why wasn't it queued and then forwarded? Was this another Linux shortcoming?

On investigation we found that backup MX (mail exchanger) records were in place to take care of this very problem; however, they were pointing to the wrong machine. Again, the problem could not be pinned on Linux; it was an administrative error by a previous systems administrator. The mistake went undetected because this backup had never been exercised before, since Linux had been working flawlessly.

Strange Software Problems

Let's move on to those *strange software problems* I mentioned. Surely we can find something to pin on Linux here.

One machine, used as our DNS name server, had been less than reliable. Two things in particular happened quite regularly. The first was that **syslogd**, the system log daemon, would hang in a loop eating up all available CPU time. While this problem appeared to be related to the location of the log file disappearing (caused by a reboot of the file server or a network problem), we haven't been able to fix it. However, it doesn't appear to happen in newer Linux releases (our problem machine is running 1.2.13) and, while it is irritating, it does not cause the machine to crash—just to run slower than normal.

The other problem on this same machine was stranger, although it turned out to be fixable. Multiple copies of **crond** (the cron daemon) kept appearing on the machine even though only one was initiated at boot time. One day, I found 13 **crond** jobs running, killed 12 and, a few hours later, found three still running.

At this point Jay jokingly said, “Maybe there is a cron job starting cron jobs.” Well, since there were processes being started by cron that didn't exist on the other machines, I started looking around for suspicious jobs. The first couple of extraneous jobs I found were benign, but then I found one that made both of us realize that Jay's attempt at a joke wasn't a joke at all. There was, in fact, a cron job that initiated another cron job. Or, more accurately, a cron job that grepped for everything but a cron job, attempted to kill all cron jobs and then started a new cron job. In other words, it looked like a partially written script to do “who knows what”--nothing that would actually work. It was signed and dated, so we could see both who wrote it and that the creation date was about the time when stability problems first appeared. Again, we had found “pilot error”, not a problem with Linux.

There are more of these stories than there is space to tell them. Basically what we found out was that even though various distributions may have some kinks in them like a wrong file permission at install time, they do all install. That's true for Caldera, Linux FT, Debian, Red Hat, Slackware, Yggdrasil and all the others. Software does not *wear out*. If the system is running it is not likely to stop aside from hardware errors. As a case in point, I still have a 0.99 kernel running on the main machine at fylz.com that was installed in August, 1993. It is an NFS server with three 38.8KB modems on it. The hardware is a 386DX40 with 8MB of RAM. Why haven't I upgraded it? It works, and it is extremely stable. The last reboot was in November 1996, when I turned off the machine to remove a zip drive from the SCSI bus.

What We Are Doing Differently

We are now back up and running and continuing our conversion to Debian. We are fixing problems as we find them—not just hardware and software but also administrative. For example, if we had been able to find the cellular phone number of our ISP, we could have had him address some problems much more quickly than by communicating via e-mail from a system that was mostly broken.

Before I get into specifics, let me say, no matter what software you are running, it is hard to program around hardware malfunctions. Detecting and halting them is about the best you can do.

You can also do backups and document. At some point we had a configuration disk for our firewall; but when we needed to replace the hard disk, the configuration disk had vanished. This loss cost hours of work time and probably a day of uptime. Having a complete backup of everything, boot disks for all machines, spare cables and disk drives and other assorted parts can make a big difference in the elapsed time to deal with a problem.

Another essential element that we are missing is a monitoring system. If something breaks at 3AM on Sunday, it may not be noticed until 8AM Monday. Being able to detect a problem immediately is crucial to getting it fixed in a timely manner. Even if it is just one person's workstation, having it fixed or replaced before the user comes to work on Monday makes for much higher effective reliability for the network.

One other innovation that we are working on is a backup file server to perform daily backups. Scripts will copy everyone's home directories over to this machine and then its disks will be written to tape. Then, if the main server fails, we can just move this machine in to take over as main.

What's Missing?

One other thing that still seems to be a problem is a reliable Auto Mount Daemon. While we have run AMD semi-successfully, it has proved to be somewhat unstable in our configuration. AMD depends on NIS, and there have been failures caused by the physical network or the yp server.

To address this problem, Jay had an idea. Note that it is still an idea not a design specification, but it is worth mentioning. Here it is in his words:

> Anyway, my idea for a new VFS-based automount would be to have a daemon which mounted a special VFS file system similar to PROC and then, through a series of system calls, populated it according to its automount maps; NIS or file or otherwise. Whenever someone descended into a directory, the kernel would send a standard Unix signal to the daemon, which would execute a system call to figure out what it had to do. This would prevent the daemon from having to use shared memory or other system V IPC, and it would allow the daemon to be ported to Berkeley OSes.

For those not familiar with it, VFS is an interface layer between the user and the actual file system being supported. This made implementation of the DOS, System V and other file systems much easier than on systems without VFS. Implementing an auto-mount file system should be relatively easy using VFS, and it could offer an integrated, reliable solution to a problem that plagues many operating systems.

One other thing that would be nice is a journaling file system (JFS). Unix System V, Release 4 has one and it is probably the one place where SVR4 is superior to Linux. As this is an editorial rather than a technical article, let's just say that JFS maintains enough information to reconstruct the exact file system structure after a crash without having to revert to the *seek out and destroy* sort of approach that fsck uses. This means a lower probability of losing anything and much quicker reboot times after a crash.

That's it from this end. If there are people out there interested in working on either of these last two projects, drop me a line at info@linuxjournal.com. Maybe we can be better than everyone after all.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

1997 Linux Expo

Jon “maddog” Hall

Issue #39, July 1997

A report on Linux Expo, April 4-5, 1997 in North Carolina.

“ Well, should we get one pitcher or two?” That was the question which began the first unofficial event of the Linux Expo Thursday night. A group of people, including Red Hat employees, some of the speakers and a tired maddog were at the Carolina Brewery in Chapel Hill, North Carolina. It was late, and I was the last person to arrive. “ Two pitchers,” I cried, “ Now what will *you* be drinking?”

The next day, Friday April 4, started early, as I had to set up the Linux International booth, as well as absorb all that was happening. The event was held in the North Carolina Biotechnology Center at Research Triangle Park. As I approached the Biotech Center, I was met with a friendly parking coordinator who reinforced the information that “ parking was scarce” , and that most people had to park at outlying lots. Fortunately Red Hat had arranged for shuttle buses from those lots and from several of the hotels. Since our car had an exhibitor's pass, we were able to park close to the Biotech Center and unload our banners, handouts and stuffed penguins.

There was a large tent outside of the building containing the “ Linux Expo Super Store” stocked with Linux books, Linux bumper stickers, T-shirts (including an excellently designed Expo shirt that said “ Expose yourself to Linux” with a front and rear view of a penguin holding open an overcoat) and other interesting souvenir items. Further to the left was an outdoor viewing area for the conference talks that (due to the excellent weather) was a favorite spot for people to view the technical talks for free, especially while playing Frisbee. A raffle was held in the registration area, and prizes were given away on an hourly basis. Having registered, attendees were given a copy of the talks as well as an event schedule.

The event was held on two floors with the exhibits spread out on both. There was another conference viewing area inside the building with TV monitors, as well as the conference auditorium itself. There was an Install Fest area (sponsored by the Washington D.C. Linux User's Group, Linux Hardware Solutions and Red Hat Software), where people brought their systems, received help with installing Red Hat's latest release, and Olaf Kirch's kernel-based NFS server was “ stress tested” at the same time. Finally, there was a food court area, where people could buy sandwiches, chips, soda and other “ software development food” .

There were fifteen vendors at the Expo, each with “ table-top” booths to display their wares. I prefer the “ pipe-and-drape” approach to trade shows rather than expensive booths, since I would rather the vendors put more money into development of the product and less into elaborate displays or floor shows with unicycle riders who juggle things. While not all Linux vendors were at Linux Expo, a wide spectrum of companies, including Linux International, Cyclades, Numerical Algorithms Group, Linux Hardware Solutions, Enhanced Software Technologies, Caldera, Applix, Xess, WorkGroup Solutions, Stay Online, VA Research, Apex Systems Integration, PromoX Systems and (of course) Red Hat Software were present. One item being demonstrated at the Linux Hardware Solutions booth was a free piece of software called **em86** that allowed an Intel/ Linux binary to run without change on an Alpha/Linux system. Being shown for the first time, it allowed Applixware, Netscape and various other applications to execute as if they had been ported to the system.

Penguins abounded in various T-shirts, giveaways and objects d'art. In fact, so many people were there (I estimated 900 over the two-day event) with penguin “ stuff” , dwthat I thought I'd had enough of penguins; but afterwards while wandering around Chapel Hill, Alan Cox found some candy in the shape of penguins, so “ penguin lust” started all over again.

The technical conference started off with a presentation by Gilbert Coville of Apple Computer with a talk about the MkLinux kernel. For people who were afraid that this would turn into a “ Red Hat Only” event, it was interesting that Gilbert's talk opened the Expo and that a talk about the Debian Linux Distribution (given by Bruce Perens) followed shortly after. Bruce also discussed the graphics used in the making of Toy Story in a separate presentation.

Various presentations about hardware-specific ports were given. Dave Miller talked about the “ Next Generation SPARCLinux” as well as the Free Software Development Model, and David Mosberger-Tang talked about the Alpha Port, as well as methods, applicable to both Intel and Alpha, for speeding up your programs by paying attention to memory and cache accesses.

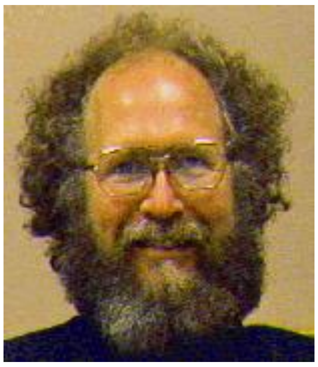
Other talks were more general across the Linux OS, such as Jeff Uphoff's “ Network File Locking” , Alan Cox's “ Tour of the Linux Networking Stack” , Peter

Braam's “ Coda Filesystem” , Alexander Yuriev's talk on the IPv4 family of protocols and infrastructure and his talk on security, Michael Callahan's “ Linux and Legacy LANs” , Eric Youngdale's “ Beyond ELF” , Olaf Kirch's “ Linux Network File System” , Theodore Ts'o's “ Ext2 File System: Design, Implementation and the Future” , Miguel de Icaza's talk on the new RAID code and Daniel Quinlan's talk on the File System Hierarchy Standard.

To round out the list of talks and events was Dr. Greg Wettstein's talk on “ Working and Playing with others: Linux Grows Up” and the Linux Bowl.

The Linux Bowl was the final event. Two teams of six developers were pitted against each other to answer thirty questions about Linux and the Linux community. Questions ranged from “ What liquid should one drink between rounds of a Finnish sauna?” (correct answer: beer) to “ What version library fixed a particular security hole?” to which Alan Cox gave a (seemingly) ten minute answer. While some of the questions were very obscure (even the moderator was unsure of the answer), most of the time either the right answer (or a good facsimile) was given.

Finally, I would like to thank the members of the Atlanta Linux Enthusiasts (http://www.ale.org/) group who helped to staff the Linux International booth. They were great and helped give me the freedom to get out from behind the booth every once in a while, because most importantly, Linux Expo was a chance to talk with the vendors, the developers and other old and new friends on a one-to-one basis. Perhaps some things could be improved for next year: A larger auditorium for the talks, more and closer parking and less expensive food in the food court. But certainly the southern hospitality and warmth of Red Hat Software came through. I want to thank the sponsors for arranging a great event, and I hope that next year's will be even larger and better.


Jon “maddog” Hall is Senior Leader of Digital UNIX Base Product Marketing, Digital Equipment Corporation.

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

wc—Word Count

Alexandre Valente Sousa

Issue #39, July 1997

A simple command with a lot of uses.

The **wc** (word count) command is a very simple utility found in all Unix variants. Its purpose is counting the number of lines, words and characters of text files. If multiple files are specified, **wc** produces a count for each file, plus totals for all files.

When used without options **wc** prints the number of lines, words and characters, in that order. A word is a sequence of one or more characters delimited by whitespace. If we want fewer than the three counts, we use options to select what is to be printed: **-l** to print lines, **-w** to print words and **-c** to print characters. The GNU version of **wc** found in Linux systems also supports the long options format: **--chars** (or **--bytes**), **--words**, **--lines**.

When I applied **wc** to an earlier version of the LaTeX source file with this text, I received the following information from **wc**:

```
wc wc.tex
     98      760     4269 wc.tex
```

This line means that the file had 98 lines, 760 words and 4269 characters (bytes). Actually, I seldom use **wc** alone. Due to its simplicity **wc** is mostly useful when used in combination with other Linux commands.

If we use a file system other than Linux (or Unix), namely DOS, there is an ambiguity due to a line break being a combination of a carriage return and a line feed. Should **-c** count a line break as two characters or only one? The POSIX.2 standard dictates that **-c** actually counts bytes, not characters, and it provides the **-m** option to count characters. This option cannot be used together with **-c**, and for that matter, GNU **wc** does not support **-m**. If we desperately need it, we can always subtract the line count from the byte count to obtain the char count of a DOS file. Here are two different ways to achieve this:

```
wc /dosc/autoexec.bat | awk '{print $3-$1}'
tr -d '\015' < /dosc/autoexec.bat | wc -c
```

The first solution uses **awk** to subtract the first field (the line count) from the third field (the byte count). The second solution uses **tr** to delete the carriage returns (char 15 in octal) from the input before feeding it to **wc**.

Recently I used a CD-ROM writer that was connected to a machine that was slightly sick. Now and then a block of 32 consecutive bytes got corrupted while copying amongst different hard disk partitions. This caused quite a few CD-ROM backups to be damaged. Sometimes the damage affected a large file, and in this case, it was cheaper to keep the bad file and add a small patch file to the next backup. To decide whether we should make a new full backup of a corrupted file or just make a differential patch, we used the **cmp** command to detect the differences, followed by **wc** to count them:.

```
cmp -l /original/foo /cdrom/foo | wc -l
```

The **-l** option to **cmp** provides a full listing of the differences, one per line, instead of stopping on the first difference. Thus, the above command outputs the number of bytes that are wrong.

If we want to count how many words are in line 70 of file **foo.txt** then we use:

```
head -70 foo.txt | tail -1 | wc -w
```

Here, the command **head -70** outputs the first 70 lines of the file, the command **tail -1** (i.e., the number 1) outputs the last line of its input, which happens to be line 70 of **foo.txt**, and **wc** counts how many words are in that line.

If our boss presses us to include in our monthly project report a count of the number of lines of code produced, then we can do it like this:

```
wc -l */*.[ch] | tail -1 | awk '{print $1}'
```

This assumes that all our code is in files with extension **.h** or **.c**, and that these files live in subdirectories one level deep from our current directory. If file depth is arbitrary, we use the following:

```
wc -l `find . -name "*.[ch]" -print` | \
         tail -1 | awk '{print $1}'
```

Notice the use of back quotes in the **find** command line, and forward (normal) quotes in the **awk** command. The command **find . -name "*.[ch]" -print** outputs the ***.c** and ***.h** files located below the current directory, one per line. The back quotes cause that command to be executed, and then replace each newline in the command's output with a blank, and pass that output to the **wc** command line.

If in good GNU style you mark all current bugs and dirty hacks in your source code with the word **FIXME**, then you can see how much urgent work is pending by typing:

```
grep FIXME *.c | wc -l
```

The **grep** outputs all lines that have a **FIXME**, and then we just have to count them.

As you can see there is nothing special about the **wc** command; however, half of my shell scripts would stop working if that command was not available.

Alexandre (avs@daimi.aau.dk) is from Porto, Denmark, but has been in Aarhus for his PhD, just delivered—something to do with literate programming and stuff. He is ashamed to confess that his first Linux was 1.02, but he is playing catch up. He claims to have brainwashed his significant other, Renata, and now she is even more sanguine about Linux. Now they are threatening to capture the mind and soul of their innocent 9 year old daughter Maria. She has a Mac but with the release of MkLinux she is no longer safe. Root password at 9? Cool.

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

MYDATA's Industrial Robots

Tom Björkholm

Issue #39, July 1997

This company is using Linux to control industrial robots—here's why.

At MYDATA automation AB (a Swedish robotics company) we have chosen Linux 2.0 as the new operating system for our pick-and-place machines. The Linux version of our control software is currently running on in-house test machines. Linux will completely replace the older “real-time Unix” on customers' machines in the third quarter of 1997.

The Pick and Place Machine

A pick-and-place machine is a special-purpose industrial robot. It is designed to pick electronic (surface mountable) components and place them in the correct position on a printed circuit board (PCB). PCBs are boards found inside electronic equipment, such as motherboard and plug-in cards found in any personal computer.

The design of a pick-and-place robot is built around a split-axis concept. We have a high speed X “wagon” (yes, we call it a wagon, despite the fact that it isn't pulled by horses) moving from left to right (the X axis) on top of the machine. On the X wagon we have a mount head, moving vertically on the Z axis and rotating on the phi axis. We also have a Y wagon, called a “table”, moving on the Y axis. (See Figure 1.)

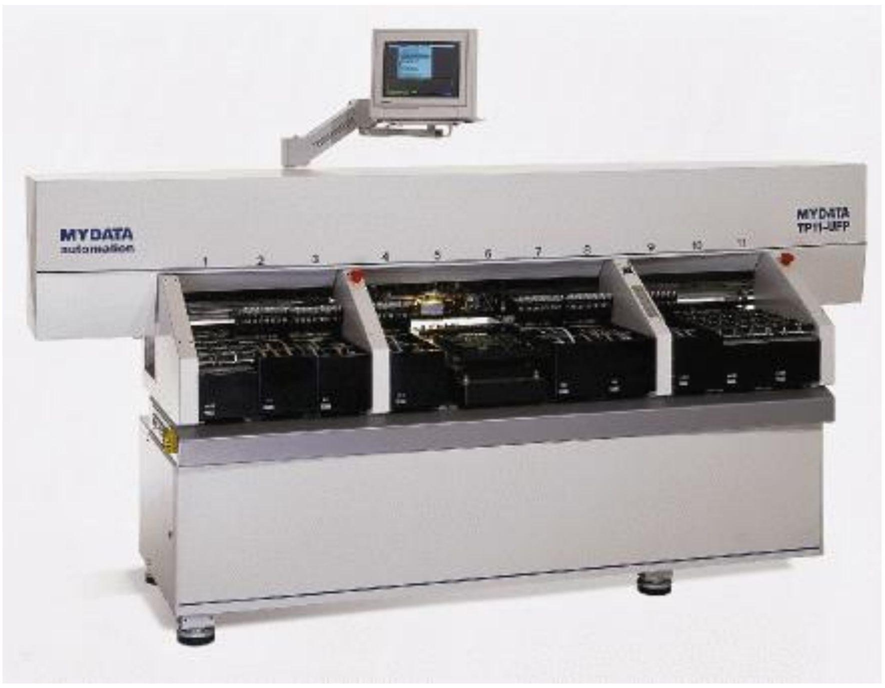


Figure 1. A MYDATA pick-and-place machine model TP11.

The components are supplied in tape reels, component sticks, or on trays which are loaded into “smart” magazines that position the component in a pick position directly under the path of the X wagon.

Let us follow a single mount cycle. First, the magazine positions a component in the pick position. Next, the X wagon moves to the position above the magazine. The mount head is lowered, and the tool tip touches the component. Vacuum is applied, and the component is sucked to the tool, much like drinking through a straw. (See Figure 2.) The mount head moves up, lifting the component, and the X wagon moves to the Y wagon.

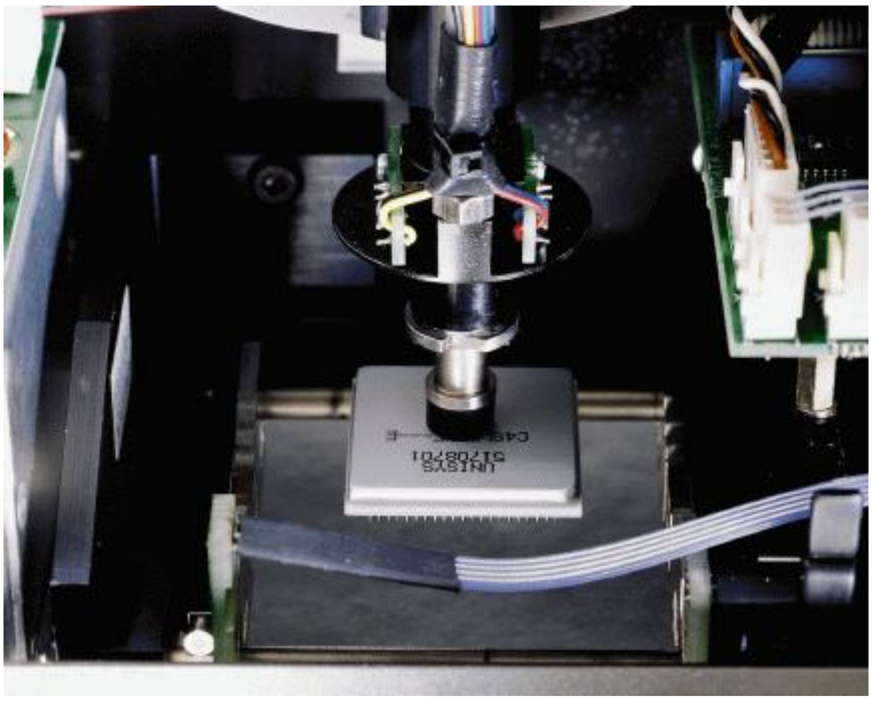


Figure 2. A mount tool applies suction to a large component. Mount tools come in different sizes for various components, which are used as needed.

As each component must be placed with a much higher precision than the carriers in which it was delivered, some means of centering the component on the tool is needed. Thus, two centering jaws push the component to the middle of the tool tip, while the dimensions of the component are measured. Then the tool and component turn 90 degrees and the centering and measuring is done in this direction, too. Finally, the electrical properties of the component are measured. (See Figure 3.)

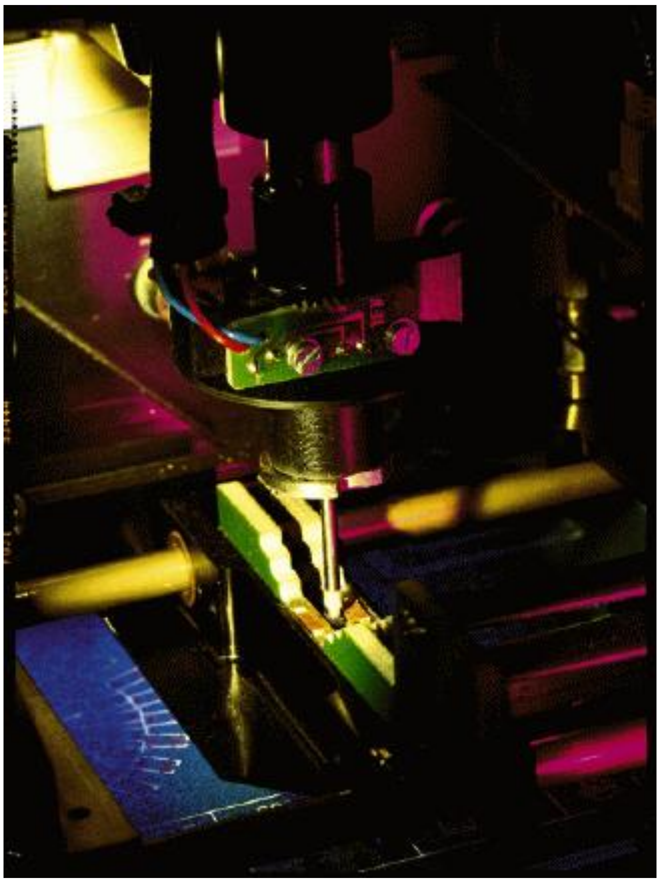


Figure 3. The centering jaws are pressed against a small component and the electrical properties of the component are measured with electrodes.

During this X movement towards the place position, the Y wagon carrying the PCB positions itself at the correct Y coordinate for the component to be placed. Finally, the mount tool lowers and places the component in the correct position. The entire mount cycle is completed in approximately half a second.

To speed up mounting many small components there is also a Hydra head on the X wagon which picks eight components simultaneously and places them individually. This device cannot use mechanical centering, and it uses optical centering instead. During optical centering the position of the component on the tool is measured optically and adjusted for when placing the component.

(See Figure 4.) In addition, the Y wagon can “dock” to a conveyor to load or unload PCBs. (See Figure 5.)

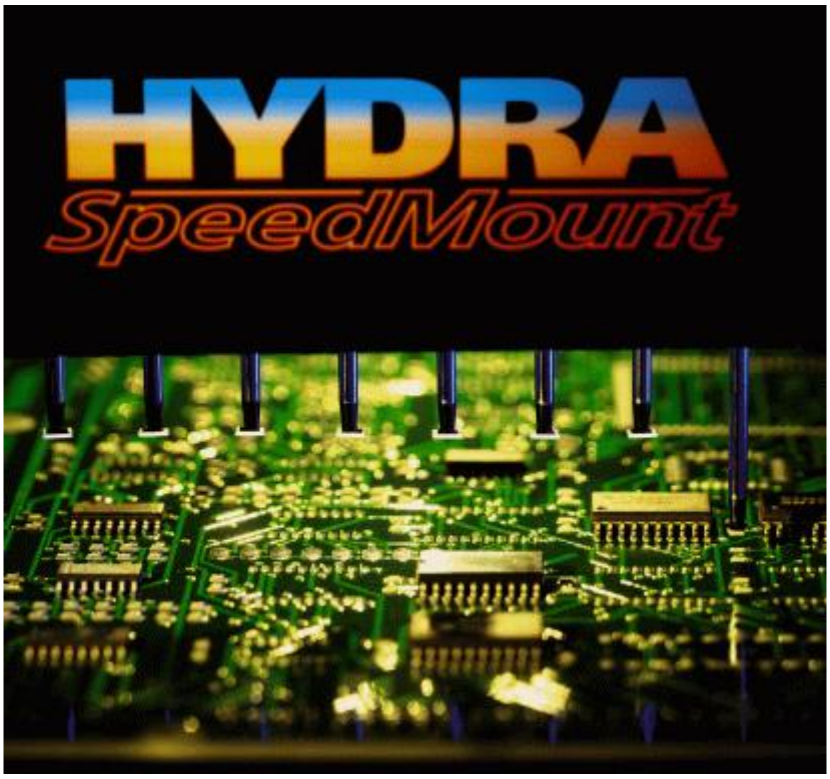


Figure 4. The Hydra head includes 8 tools.

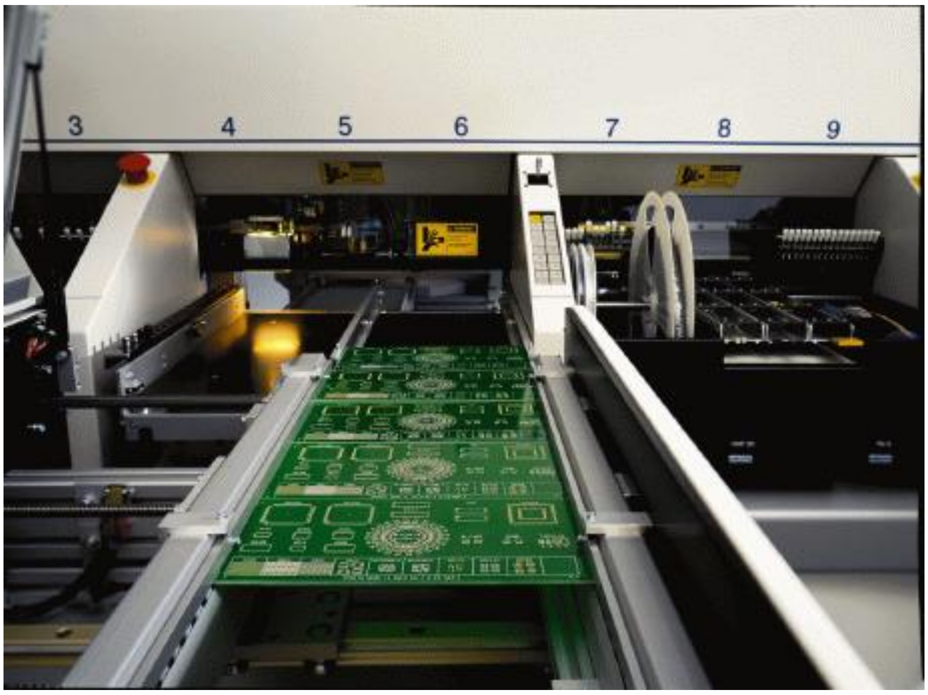


Figure 5. The Y wagon has docked to the panel conveyer VPC to load PCBs.

Hard and Soft Real-Time

Long ago MYDATA made the smart design decision to separate the hard real-time requirements from the complex algorithms of optimization, user interface and database management. In a typical MYDATA machine, there is a computer box containing one Pentium PC motherboard and several servo computers. (See Figure 6.) We call the servo system "hard real-time" and the Un*x system "soft real-time".

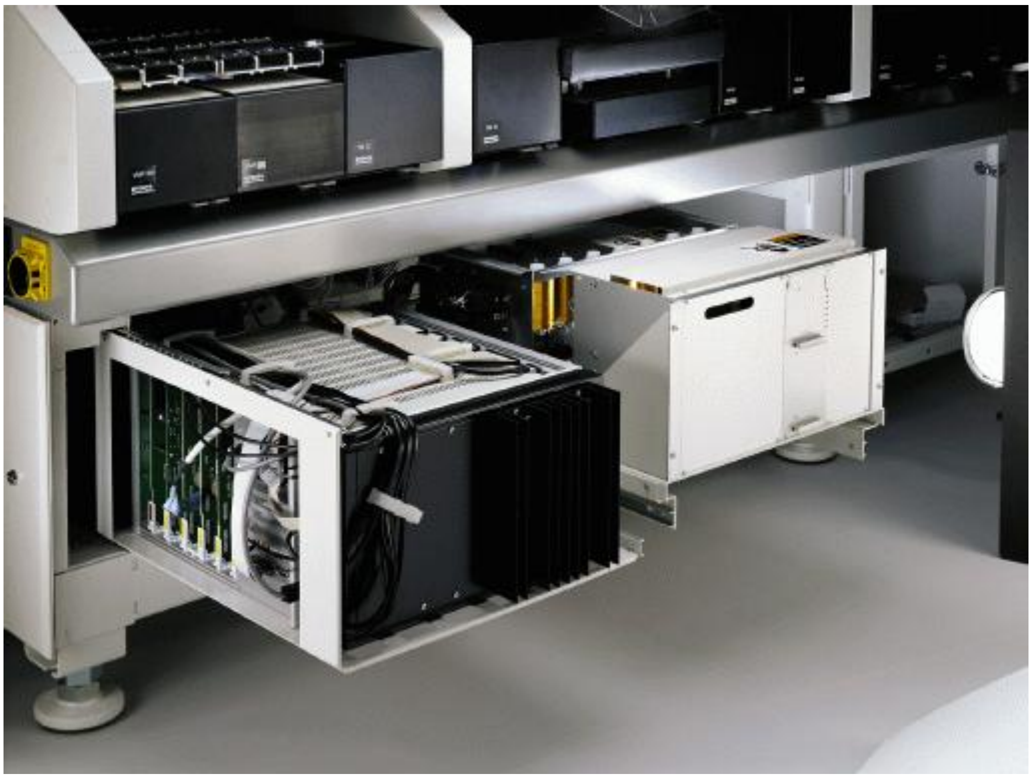


Figure 6. A MYDATA machine with the front cover removed. The computer box is in the left, revealing several servo computers. The Pentium motherboard is hidden in front. The box on the right contains battery backup and other electronic components.

The servo computers are programmed in Z80 assembler and take such commands as "move to position 2500 with acceleration parameters XA XB XC". The positions given are in ticks on the sensors.

The PC motherboard currently runs Un*x—a true multitasking system running several programs written in C++. One of the programs is a mounter process that generates the commands sent to the servo computers specifying where to mount each component, and also handles coordinate system transformations and conversion to sensor "ticks". The optimizer process optimizes the mount order for maximum throughput. The data server daemon stores information about packages, components, PCBs and component locations. There is also a Man-Machine-Interface process to communicate with the user.

In the servo programs accurate timing is a must. However, in the Un*x system a short delay will not break anything, but will result in poor overall mounting speed. Thus, what is needed for the Un*x system is not a guaranteed response time, but a very fast average response time.

Guerilla Tests and Lobbying

When I joined MYDATA, I had been using Linux for years. I thought that programming Unix on PC hardware would be comparable to running Linux. MYDATA was then using Venix 3.2, an "industrial strength real-time operating system" from VentureCom based on System V, release 3.2. After a short time of programming Venix I was very disappointed. None of the tools that I usually relied on were available: no emacs, no bash, not even a **strerror** function in the C library.

In the summer of 1995, I began lobbying for a more modern operating system. There were two alternatives: Venix 4.2 (supported mostly by old timers), and Linux 1.2.8 (supported by a few young programmers). A change to either of these operating systems meant a massive porting effort.

After a lot of nagging I got permission to test the interrupt response time for both operating systems under different load conditions. The test was done with a special plug-in card with output ports and interrupt generation connected to an oscilloscope.

Everyone expected this test to kill Linux as an alternative. In Venix I had a hard real-time process, and in Linux I used the **nice** levels to create a high priority process. The surprising result was that the average interrupt response time was three times faster on Linux—and the worst case interrupt response time was ten times faster on Linux.

Despite the great test results, there was not much interest in Linux. A few programmers wanted to use Linux, but the marketing department had invested a lot in Venix as the "industrial strength real time OS". There was also strong resistance from people who were unwilling to learn a new operating system.

Then, in December 1995, circumstances changed. 3Com could no longer deliver the 3c503 Ethernet card that Venix 3.2 used. MYDATA managed to find an old stock of WD8003 cards with device drivers for Venix, and I got permission to start a one man project doing a pilot port of the system to Linux.

The Pilot Port

An industrial robot such as the MYDATA pick-and-place machine has a lot of special hardware. My first job was to rewrite all the device drivers for Linux, which was interesting, produced no surprises—and was completed on schedule. Of course, the fact that I had source code for the Venix device drivers was a distinct advantage.

In early 1996 I learned that Markus Kuhn (a German student) had added POSIX real-time scheduling to Linux 1.3. This was a gift from heaven—hard real-time priorities and memory locking in Linux.

During the spring I worked on porting the application programs. This work proved to be a lot more difficult than I had expected. I would get a program to compile, just to have it immediately crash when running. After long hours of debugging I found an uninitialized variable. Due to different memory layouts, the program just happened to work in Venix. With the bug fixed I would run the program again, and again it would crash for similar reasons.

Another cause of great grief was using C++ objects in file scope and in global scope. If you declare two C++ variable in file scope, the constructors for those variables will be run before **main** is started. However, if the variables are declared in different files, the order in which the constructors are run will be defined by the implementation. It turned out that the Venix linker ordered the constructors in alphabetical order of object file names, whereas the Linux linker ordered them in the sequence the object files were added to the library using **ar** (archive).

The result was that suddenly we had a lot of code which contained uninitialized variables. After fighting with the code to solve these problems I am firmly convinced that both file scope and global variables should be banned from C++.

As spring grew into summer I continued in this manner with welcome help from another programmer, Carl Martinsson. Finally, in June 1996, our joint effort was rewarded. We had a MYDATA pick-and-place machine running Linux and actually mounting components on a PCB.

The main lesson I learned from this porting effort is that moving non-portable Unix programs from one Unix-like OS to another is much harder than one might think. Also, rewriting device drivers for Linux is not that hard.

Real Linux Version

During the work on the port to Linux, the software team developed a new version of the software (with new features) for Venix. In October 1996 we finally received the decision that the next release of the software was to run under Linux.

With the help of our code-management system from Perforce (http://www.perforce.com/), we managed to merge the newly developed features for the Venix system with the Linux system. Of course, this resulted in another round of debugging.

Now, in early 1997, the Linux version of our software is the main development environment and is fully functional. Support hardware currently being developed needs to be completed before we can start the verification tests and ship the software to beta test sites. The release of the Linux-based system to customers worldwide is scheduled for the third quarter of 1997.

The MYDATA pick-and-place software consists of 8,000 files and 150MB of source code. MYDATA is shipping roughly one pick-and-place machine a day at $150,000 each. I think this is the first project to utilize Linux as a real-time platform on this scale.

Drivers for the Free World

Most of the hardware in the MYDATA machines is manufactured in-house. This hardware is impossible for the average person to buy and would be useless unless he had a MYDATA machine. For this reason, we have not made the device drivers publicly available. In any case, MYDATA hardware is already bundled with the most up-to-date software available.

We do, however, use two pieces of hardware that can be used by anyone: the Decision PCCOM-8 multi port serial card and the parallel port to SCSI conversion cable from Shuttle Technologies called EPST. We contracted Signum Support, a Swedish consulting company specializing in GNU software, to write device drivers for these two. The contract states the device drivers are to have the GNU GPL license and that Signum should try to include the drivers in Linus Torvalds' next kernel release. These drivers are currently available and fully functional. Contact Signum Support at http://www.signum.se/ for download information.

From a purely economic point of view, we get a lot of beta testers for these device drivers free of charge. Even better, we get to give something back to the Linux community.

Installation Procedure

From a programmer's and lawyer's point of view, MYDATA supplies four different software packages that interact.

1. The Linux operating system (free of charge)
2. The device drivers for MYDATA hardware (as loadable modules)
3. Application programs
4. Servo programs (running on servo computers)

However, from the user's point of view MYDATA offers an industrial robot with hardware and a software system. The user is not usually concerned with implementation details such as the choice of an operating system.

Until now, the software installation and upgrades have always been done by service engineers. With the next generation of software we want to provide an installation CD-ROM which the user can use without a service engineer or thick manual by his side. The vision is that the user will insert the CD-ROM and the boot floppy, turn on the power and let the installation process automatically. Well, we might have to ask for an IP address. This installation CD-ROM is being developed in cooperation with Signum Support and is based on Red Hat 4.0 with a limited choice of hardware and added hardware detection.

Conclusions

MYDATA has managed to switch from an aging operating system to Linux with a reasonable amount of effort. In Linux we now have a modern operating system that will continue to grow with us for several years, and it gives us a solid foundation for future development.

Tom Björkholm is a 32 year old software engineer. He has used Linux since version 0.95. When not programming, he enjoys sailing. By the time you read this article, he will have left MYDATA to work for Ericsson Business Networks. He welcomes comments sent to Tom.Bjorkholm@ebc.ericsson.se, or by snail mail c/o *Linux Journal*.

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

CLUELESS at the Prompt

Mike List

Issue #39, July 1997

Here are some tips for the novice in our newest column designed for the newcomer to Linux.

Multitasking

Welcome to installment 3 of Clueless at the Prompt. Thanks for the encouraging e-mail. In response to several requests, here is a little information to help you get your feet wet.

If you are familiar with that other windowing system, you may be aware of the concept of multitasking. Using a single computer to do several applications at once is a highly desirable trait for an operating system (OS).

It's fairly obvious how to accomplish this in a windowing environment, but not so obvious at the shell prompt. Here are some of the details.

When you start a program at the shell prompt, you can stop it by typing:

```
Ctrl-Z
```

which returns you to the shell prompt. Then type:

```
bg
```

and the program or job is restarted in the (b)ack(g)round. A job can also be put in the background from the beginning by typing an ampersand (&) at the end of the command, e.g., **make filename&**. Running your job in the background allows you to run another job at the same time without changing to a different virtual console (VC). The job can be brought back to the (f)ore(g)round by typing: **fg**, if its the only background job, or **fg *`jobno`***, if there is more than one.

You can change to a different VC by using the Alt-F2 (through F6) command. Each one of these can also be used in the manner that I have described, to the extent that you can easily run out of resources in a fit of deep-hack-mode euphoria, if you aren't careful. If you get really exuberant, you could even forget which jobs you have running. Relax, you can “remember” them by typing:

```
jobs
```

This command lists all jobs running in the background, much like the **ps** command lists all processes that are currently using up your precious memory and CPU.

Mount

When you boot up Linux, your file system or rather your hard drive must be mounted, so that the file system can be read and acted on. Your floppy drive, tape backup or CD-ROM cannot be automatically mounted, so you may need the **mount** utility. For example:

```
mount -t ext2 /dev/fd0 /mnt
```

or:

```
mount -t msdos /dev/fd0 /mnt
```

mounts your floppy drive (called a: by DOS) to a directory called /mnt from which you can access files on floppy disks. In the first example, the /mnt directory can be read into the ext2 file system, while the second reads floppies written in MS-DOS format. To read the contents of the floppy drive, which is now /mnt you can type:

```
cd /mnt
```

then, **ls** or **less *`filename`***.

In a similar manner, you can mount your other floppy drives, tape drives, CD-ROMs or other read/write devices. These devices can be unmounted using the command:

```
umount /dev/fd0
```

or **umount /dev/*whateveryoumounted*<\n>**.

Some Timesavers…

Here are a few tips that can make your Linux life a little easier.

Suppose you make a typo in your command that you don't catch until after you press enter. If the environment variable **FCEDIT** is set to **emacs** (default for the bash shell), you can correct it without retyping the whole command by tapping the up arrow key to bring back the previous command for editing with **emacs**. In fact, you can use the up arrow key to go up through the history file and the down arrow to come back down if you go too far. If **FCEDIT** is set to **vi** (default for the ksh shell), press ESC (escape key) then use **k** to go up the history file and **j** to go down, and of course, use **vi** to edit the faulty command.

To change back to a directory you have just left or to scan subdirectories, use:

```
c -
```

in the following manner. Change from your /home directory to the main trunk directory:

```
cd /
```

then, to look at the top level of each directory, for instance, type:

```
cd usr
```

then **ls**. If you didn't find what you were looking for, type:

cd -

and you will find yourself at the trunk / again. Unfortunately, you can only go one layer deep, but it is still useful when you install a source package and want to check out the contents of each of the subdirectories.

Sometimes, at least at first, you may not know how to stop a program or process that's running, but you are unwilling to let it slowly eat up your memory or CPU overhead. To get rid of it, first find its pid (process identification) number by typing:

```
ps  -a
```

to get a list of all running processes. Make note of the pid number and type:

```
kill
```

There is another way to do this that is actually easier. Browse through the LSM (Linux Software Map) for a utility, actually a nicety, called die-1.1. You can unpack this utility into a directory or use **installpkg dopkg** or whatever your single package installation utility is. Then look for the /die-1.1 and **cd** to it. This directory contains a couple of files: a source file called die11.c and a documentation file called die.doc.

Assuming that you have installed the GCC compiler, just type:

```
gcc -o die die11.c
```

hit ENTER, and presto, you've compiled the **die** utility. Just **mv** it to a directory in your path, and if you like, **mv** the die .doc to /usr/doc or somewhere it can be with other help text friends (but not man pages, they'll pick on it unmercifully). Now, next time you're in a quandary about how to gun down a process just type:

```
die
```

and it will do the deed without having to look up the pid number. To find out more about **die** just type:

```
die
```

with no argument and it will give you a summary of the commands you can try the up arrow keys on.

Disclaimer

You have probably noticed that this column is shorter than the previous two, presumably since Linux is really an easy OS to learn so my curve isn't as steep; or maybe it's the fact that I have gone half crazy trying to install a DECvt220 to a serial port and it refuses to cooperate.

Next Time—let me know what you would like to see in this column, and I'll try to oblige. Just e-mail (troll@net-link.net) me and ask; otherwise, I'll just write about what has given me trouble and how I got past it.

Mike List is a father of four teenagers, musician, printer (not laser jet), and recently reformed technophobe, who has been into computers since April, 1996, and Linux since July. He can be reached via e-mail at troll@net-link.net.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

New Products

Amy Kukuk

Issue #39, July 1997

AcceleratedX Server Version 3.1, CDE Business Desktop, DNEWS News Server 4.0 and more.

AcceleratedX Server Version 3.1

Xi Graphics (formerly X Inside) announced the release of AcceleratedX Server Version 3.1 for Linux. AcceleratedX is an X Windows display server that solves compatibility and correctness problems caused by the lack of available driver support for new hardware. It features multiple visuals, overlays and gamma correction for industry-standard PCs. AcceleratedX supports graphics hardware from ATI, S3, Matrox, Diamond and more.

Contact:Xi Graphics 1801, Broadway Suite 1710, Denver, CO 80202, Phone: 303-298-7478, Fax: 303-298-1406, E-mail: info@xig.com, URL: http://www.xig.com/.

CDE Business Desktop

Xi Graphics also announced the release of CDE Business Desktop, a complete networked operating system that enables system administrators to integrate industry-standard PCs into existing Unix networks with the same functionality as more expensive systems. CDE provides users with a consistent graphical user interface and incorporates several enhancements, including the latest AcceleratedX Server, OSF-certified Motif + Development, an industry-standard Linux distribution and a comprehensive internationalization package which includes all major European languages and Japanese. CDE allows users to customize their desktops to individual work styles and preferences without sacrificing administrative control over system configuration. It also provides a common method of installing and integrating applications in a distributed multi-user environment. For more information visit the company's web site at: http://www.xig.com/.

Contact:Xi Graphics 1801, Broadway Suite 1710, Denver, CO 80202, Phone: 303-298-7478, Fax: 303-298-1406, E-mail: info@xig.com, URL: http://www.xig.com/.

DNEWS News Server 4.0

Netwin Ltd. announced the availability of DNEWS News Server 4.0. The database has been completely redesigned, and it will no longer use a group based directory structure. Additions since the last version include; increased speed, ability to spread the database over a number of disks and automatic recovery of database integrity even after a power cut.

Contact: Netwin Ltd., PO Box 27574, Auckland, New Zealand, Phone: +64 9 6300 689, E-mail: netwin@netwinsite.com, URL: http://netwinsite.com/.

SecureNet PRO v2.0

MimeStar, Inc. announced the availability of SecureNet PRO v2.0. SecureNet's intrusion detection and response capabilities allow for the protection of entire networks from one centralized location. It automatically monitors in real-time all activity taking place on the network, protecting against computer hackers, software pirates and electronic vandals. SecureNet PRO combines securities technologies into one package. The price of SecureNet PRO 2.0 depends upon the type of license purchased. A class C license which includes the SecureNet PRO server, one client and a printed/bound manual is $1,400.

Contact: MimeStar, Inc., PO Box 11648, Blacksburg, VA, 24062-1648, E-mail: mimestar@mimestar.com, URL: http://www.MimeStar.com/.

Dynamic Modules

Automath announced Dynamic Modules. It is now possible t link existing applications or runtime-libraries to a MuPAD system. This is not realized by using network protocols, files or message-passing but by direct linking of the application program to the MuPAD kernel. MuPAD and the external program then run in the same memory space and can exchange date rapidly and efficiently. Modules are created with the MuPAD Module-Generator and the accompanying toolbox which simplifies and streamlines the access to all functions of the MuPAD kernel.

Contact:Automath, Warburger Strase 100 D 33098, Paderborn, Germany, Phone: 05251-69-2627, Fax: 05251-60-3836, E-mail: benno@uni-paderborn.de.

Stronghold Encrypting SSL Web Server

Berkeley Software Design, Inc. and C2Net Software, Inc. announced that BSDI will market and support C2Net's Stronghold encrypting SSL Web Server. Stronghold is the first server to provide full-strength cryptography to the world because it is not restricted by export restrictions. BSDI will include a 60-day trial version of Stronghold with its BSDI Internet Server Version 3.0. The Stronghold Web server is available from BSDI for $495. It is available immediately via BSDI's Web site at http://www.bsdi.com/. Contact:Berkley Software Design, 1212 Broadway Oakland, CA 94612, Phone: 510-986-8770, Fax: 510-986-8777, E-mail: info@bsdi.com, URL: http://www.c2.net/.

SOLID Desktop

Information Technology Ltd announced a campaign targeted at the community of Linux developers. Between March and September 1997 Linux enthusiasts will be presented with a free personal version of the SQL database engine SOLID Server. Numerous commercial and non-commercial web sites are joining Solid Information Technology Ltd in this program by letting their web visitors download the free SOLID Desktop from their site. The SOLID Desktop for Linux is offered free of charge for anyone to download from numerous web sites around the world. The license is for personal and development use.

Contact:Solid Information Technology Ltd, Huovitie 3, FIN-00400 Helsinki, Finland, Phone: +358-9-477-4730, Fax: +358-9-477-47-390, E-mail: iko.rein@solidtech.com, URL: http://www.solidtech.com/.

Lone Star LONE-TAR 2.2

Lone Star Software Corporation (Cactus International) announced the release of LONE-TAR Version 2.2. Lone Star has added new features to LONE-TAR in the new version including: addition of TCP/IP network support, 30% faster Bit-Level with the ability to verify the integrity of symbolic links and report any broken links, a new restart feature that allows the user to restart a backup at a specified filename, and the ability to automatically exclude any network mounted file systems and “read-only” mounted file systems such as a CD-ROM. LONE-TAR includes a standard one year technical support.

Contact: Lone Star Software, 13987 West Annapolis CT, Mount Airy, MD 21771, Phone: 301-829-1622, Fax: 301-829-1623.

Cactus System Crash AIR-BAG 3.4.1.1

Cactus International (Lone Star) announced the release of Cactus System Crash AIR-BAG 3.4.1.1. Among the new features added are: a Modem-dial-in facility with file transfer abilities, Boot Time Loadable Driver support, unlimited number of hard drive support, unlimited filesystem support, and improved compacting of the Unix kernel. The system comes with a standard one year technical support.

Contact: Cactus International Inc., 13987 West Annapolis CT, Mount Airy, MD, Phone: 301-829-1622, Fax: 301-829-1623.

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.

Best of Technical Support

Various

Issue #39, July 1997

Our experts answer your technical questions.

Changing Finger Information

How can I change the “user's name” in my finger information to a different name? For instance, how could I change my name from “Dave” to “David” ? What commands would I need to use? —David Innes

You can change that information (and more) using chfn (for **ch**ange **f**inger **n**ame). To just change the user's full name, use:

```
chfn -f "Full Name" username
```

For instance, to change the full name of the user “random” , you might enter:

```
chfn -f "J. Random Hacker" random
```

changing the full name of the user “random” to “J. Random Hacker” .

If you run **chfn** as a regular user, you can leave out your own username since **chfn** defaults to changing your own information. (Only root can change another user's information.)

If you run **chfn** without the **-f** argument, it will prompt you for all the fields it can change. —Steven Pritchard President Southern Illinois Linux Users Group steve@silug.org

Release Numbers

My distribution kernel is of version 2.0.27-5. What does the last number “5” mean? Some patch sublevel? Can I patch this kernel with patch-2.0.28 and patch-2.0.29 that are generally distributed at Linux web sites? Is this kernel somewhat modified and incompatible with these patches? —Oleg Zhirov Red Hat 4.1

The “-5” is a Red Hat-specific release number, For your particular version, it is the fifth time we've had to build the package for one reason or another. It has nothing to do with the kernel version or kernel patches.

This kernel should be patchable using the standard patches. However, I usually still suggest grabbing the entire tarball of the new version that you want, since there have been problems in the past with applying patches rather than using the full source. —Donnie Barnes Red Hat Software redhat@redhat.com

Leaving Processes Running

How can a non-root user leave a process running while not logged in? I have seen this done by detaching screens under various Unix flavors, yet I haven't found a similar program for Linux. —R. J. Rodd Slackware

Most of the Linux distributions come with the program **screen** which allows you to do just what you describe. If you don't have it, you can get the source for **screen** from: FTP://prep.ai.mit.edu/pub/gnu/screen-3.7.2.tar.gz

It compiles on Linux easily. —Steven Pritchard President Southern Illinois Linux Users Group steve@silug.org

If you want to keep processes running in the background after you log out, you can employ the **nohup** command. Check the man page for details, but briefly, just insert **nohup** at the beginning of the command line you want to live on after you've logged out; e.g., if you've got a book you need to typset with LaTeX:

```
nohup latex book.tex &
```

Be sure to include the ampersand at the end to indicate to the system the process is to be run in the background. —Gary Moore, Technical Editor

Setting Up a Netware Network

I would like to place a Linux system on a network populated with IPX/SPX Netware file servers. I am hoping there is a way to give the Linux system a “Netware/IPX/SPX” personality, thus allowing access to it from existing Netware clients. Where can I get help? —Todd Morris Slackware 2.0.x

Take a look at the IPX HOWTO. Everything about IPX-based problems and solutions is explained in this document. —Pierre Ficheux Lectra Systemes

Changing X Managers

How do I make AfterStep the default window manager for X instead of fvwm2? —Mohammed Rizal Othman SuSE 4.4

For system-wide changes, edit the global **xinitrc** (usually in /usr/X11/lib/X11/ xinit/). To change your personal setup, edit the file **.xinitrc** (in your home directory) or, if you don't have an .xinitrc file, just copy the global xinitrc to ~/.xinitrc and edit that. —Steven Pritchard President Southern Illinois Linux Users Group steve@silug.org

Printing PostScript

I have been trying to create an entry in /etc/printcap for an HP Laser printer that is connected to the network using an HP Jetdirect card. I can get it to print, but I cannot get it to use the magic filter that comes with Red Hat to enable printing of PostScript files. Any help would be appreciated. —Pat Rooney Red Hat 4.1

Unfortunately, the normal **lpd** that we ship cannot handle using filters on remote printers. Only local printers may be filtered.

If you require this functionality, you can remove the **lpd** package and install **LPRng** or one of the other print servers available from the Internet. Red Hat is investigating the use of other print servers in future versions. —Donnie Barnes Red Hat Software redhat@redhat.com

NFS Version Differences

We observed a strange difference between version 1.2.1 and version 2.0.2 in mounting file systems via NFS. In the earlier version, (v 1.2.1) mounting a file system via NFS is easy and works perfectly, and in the later, we have big trouble. —Gerard Rozsavolgy Slackware 2.02

Basically, NFS used to be broken on Linux, but it is no longer. The latest kernel versions and NFS daemons should all work well together. You should seriously consider upgrading all the machines in question. In addition to the major bug fixes, NFS is a lot faster now. —Steven Pritchard President Southern Illinois Linux Users Group steve@silug.org

On-line Mail Processing For Red Hat?

I have heard that IMAP supports on-line mail processing. Is there an IMAP for Red Hat? Where can I get it? —Nga Nguyen

Red Hat has shipped the IMAP package for quite some time. It should be in the **imap** RPM on the CD or FTP site from which you installed Red Hat. You can use **rpm** or **glint** to install the package, and it should work out of the box. —Donnie Barnes Red Hat Software redhat@redhat.com

Driver Development Documentation?

I'm interested in writing drivers for a device not yet supported under Linux. Where can I find books or documentation on Linux device driver development? —Sury Slackware 1.3.20

Your best sources for information are probably the Linux kernel source and the **linux-kernel** mailing list (to which you can subscribe by sending e-mail to majordomo@vger.rutgers.edu with “subscribe linux-kernel” in the message body. —Steven Pritchar President Southern Illinois Linux Users Group steve@silug.org

Archive Index Issue Table of Contents

Advanced search

Copyright © 1994 - 2019 *Linux Journal*. All rights reserved.