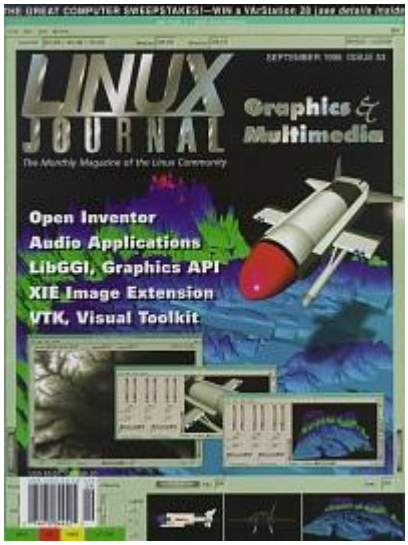
LINUX JOURNAL

Advanced search

Linux Journal Issue #53/September 1998



Features

Developing Imaging Applications with XIE *by Syd Logan*
: Mr. Logan describes the X Image Extension and show us how to use it—for the experienced C programmer.

Open Inventor *by Robert Hartley*
: Mr. Hartley shows how to do interactive 3-D programming using Open Inventor, Release 2, which he used to create the images on our cover.

LibGGI: Yet Another Graphics API *by Andreas Beck*
: The next generation fully portable graphics library

Porting SGI Audio Applications to Linux *by David Phillips and Richard Kent*
: This article describes the process of porting a variety of audio applications from the SGI platform to the Linux operating system.

Visualizing with VTK *by James C. Moore*
: A look at a new tool for visualizations of scientific data—VTK, an object-oriented visual toolkit.

News & Articles

Porting MS-DOS Graphics Applications *by Jawed Karim*
: Are you hesitant about porting your favorite VGA MS-DOS program to Linux? Using this tutorial and SVGALIB, porting will truly become a matter of minutes.

A Tale of DXPC: Differential X Protocol Compression *by Justin Gaither*

Developing Imaging Applications with XIE

Syd Logan

Issue #53, September 1998

Mr. Logan describes the X Image Extension and show us how to use it—for the experienced C programmer.

In this article I'll introduce the X Image Extension (XIE), and illustrate how it might be used by a C programmer to add image display support to a simple application. The following assumptions are made about the reader:

- You are comfortable reading C language code.
- You are comfortable with (or at least have a basic understanding of) Xlib development, e.g., creating windows, drawing text and graphics. Since the example program uses Motif, some exposure to Xt and a widget set such as Motif or Xaw would also be helpful, but is not required.

If you are new to X development, there are plenty of books that can be used to get you started. Regarding X and Motif, my favorites include *Introduction to the X Window System* by Oliver Jones (Prentice Hall), and the O'Reilly X Window series. Volumes 1 and 2 of the O'Reilly series deal with Xlib. Volume 6 deals with Motif.

Linux and XIE

I've been using XIE and Linux together since late 1994. At that time, I had to build my own X server (an early beta version of X11R6) to support XIE, plus I had to port the client library (libXIE.a) to Linux since an X environment that supported XIE was not yet available in any of the Linux distributions. Now, all currently available Linux distributions provide a more than adequate platform for XIE client development, as well as the runtime support needed for clients that use XIE.

What is XIE?

In X, a client (i.e., an application) connects to an X server, which you can think of as essentially nothing more than a display with a keyboard and a mouse attached to it. The core X protocol provides all the functionality needed by a client to produce a user interface on the server. Using the core X protocol, a client can:

- Create, move and destroy windows.
- Render graphics and text into windows and off-screen pixmaps.
- Receive notification of events occurring on the server which are of interest to the client. Such events include button presses, mouse movement, keyboard presses, window exposure notification and so forth.

For most of you, the client and server are both running on the same machine, i.e., on a single Linux system, but it doesn't have to be that way. The X Window System protocol is designed so that communication between the client and server can be carried out over a network connection, e.g., TCP/IP. (If run locally, it is a local, i.e., AF_UNIX, connection.) Therefore, you can have the situation as illustrated in Figure 1. There, the X server is my Linux machine at home. The clock application and terminal emulator (xterm) are processes running on my Linux machine locally. In addition, I have a PPP connection to my ISP (which is running Solaris or some other Linux-wannabe) and within the xterm I am executing a **telnet** session between my local machine and the ISP. The other two windows (**xiegen**) display an application executing remotely on my ISP's machine. This application is displaying its client windows on my Linux machine, responding to keyboard and mouse events that occur on my Linux machine, courtesy of the X protocol. Prior to executing the remote application, I needed to set the DISPLAY environment variable to the IP address of my Linux machine which acts as the server. Xlib (on the remote host) reads this variable on client startup and uses its value to open a connection to the X server running on my Linux machine. The :0 characters in the following line indicate a logical display on the server:

```
$ typeset -x DISPLAY=123.45.67.89:0
```

Figure 1. Linux X Server Displaying Both Local and Remote Applications

Note that except for the screen, mouse and keyboard, the client generating xiegen's output in Figure 1 is interacting with resources on the remote Solaris host. If my remote client opens a file, /etc/passwd for example, it is opening /etc/passwd on the Solaris host, not the Linux host.

In reality, running console-based UNIX or Linux applications from a dumb terminal over an RS-232 connection has much in common with running UNIX or

Linux applications from an X server over a network connection, except that when using X, the graphics support is much better.

Additional functionality can be added by vendors to the core X protocol via extension protocols. XIE is one example. Other extension protocols include the Phigs graphics extension to X (PEX), and the shape extension, which allows X to display non-rectangular windows. There are many other extensions; execute the **xdpyinfo** command to take a look at which ones are supported by your X server.

XIE is an extension that was released with the first version of X11R6 back in July 1994. XIE was developed in an attempt to provide clients with support in the following areas:

- Transmission of image data between the client and the server (in either direction)
- Image enhancement and manipulation
- Image display

The core X protocol provides only minimal support for the transmission, manipulation and display of image data. Let's look at each of these areas in more detail, and discuss what core X is missing with regard to imaging support, and what XIE brings to the table.

Image Transmission

In order to display image data, the client must first transmit the image from the client to the server. Encoding of image data, as well as efficiency of the transfer, are the two main concerns addressed by XIE.

Core X requires clients to transmit image data using X-specific encoding. If a client is working with JPEG image data, for example, it must decode the JPEG image data and convert it to X-specific encoding before sending it to the server. XIE, on the other hand, is capable of receiving and decoding image data encoded in several popular image encodings, including JPEG Baseline and the CCITT FAX encodings G31D, G32D and G42D. However, the list of encodings supported by XIE may seem fairly restricted to some; GIF is not supported because LZW has licensing issues associated with it, and PNG was not invented prior to the latest release of XIE. Vendors are free to add to the list of supported encodings. However, truly portable applications will support only encodings defined by the protocol specification. XIE supports two encodings, UncompressedSingle and UncompressedTriple, which can be used to transmit uncompressed two-tone, gray-scale and color images. Clients can use these encodings to send “raw” image data, or they can convert image data from an

unsupported encoding (e.g., LZW) to either of UncompressedSingle or UncompressedTriple prior to transmission.

In terms of efficiency, an 8-bit 640x480 gray-scale image is around 2.3MB in size, and a color version (24-bit, 640x480) is three times as large. Transmitting such large amounts of data is expensive. Because XIE allows images to be transmitted in an encoded form (e.g., JPEG), performance is increased, since fewer bytes are sent between the client and server.

Once in the server, image data can be cached in a local image storage resource called a photomap. Doing so considerably reduces the use of network bandwidth in interactive imaging applications.

Image Enhancement and Manipulation

In core X, client manipulation of image data is performed at a very low level: pixel values can only be read from or written to an image. Higher-level operations, such as image scaling, image arithmetic and blending must be implemented by the client utilizing these primitives. In addition, all manipulations must be performed on the client side, with the resulting pixel values transferred across the network from the client to the server after the manipulation has been performed.

In XIE, image manipulation is performed entirely on the server side. If XIE doesn't support a needed operation, it can be done on the client side. XIE supports high-level operations that allow the client to:

- Improve the quality of the image data.
- Perform geometric operations such as scaling, axis-flipping and rotation.
- Prepare image data for display in a window belonging to a specific X Visual class.

Interactive imaging applications are best implemented using a combination of server-side processing and image caching with photomap resources.

Photoflos

The operations to be performed on image data, including image decode, image manipulation and enhancement, and image display, are described by a data structure called a photoflo. A photoflo is a directed acyclic graph (meaning it can have no cycles) that consists of photoflo elements. Each photoflo element in a photoflo graph performs a specific atomic operation, passing its result to elements further downstream. Elements at the head of the photoflo are known as import elements, and are used to read and decode image data sent to the photoflo by the client. They also read image data directly from a photomap

resource if needed. Elements further downstream, called process elements, perform image manipulation tasks. Export elements are used to route the image data to a window, a photomap resource or back to the client and are found at the end of a photoflo graph.

Figure 2. Photoflo Graph

Figure 2 illustrates a simple photoflo graph, one we will use later in our example. The first photoflo consists of two elements, **ImportClientPhoto** (ICP) and **ExportPhotomap** (EP). ImportClientPhoto is used here to read and decode a JPEG image. ExportPhotomap reads the result from ImportClientPhoto and stores it in a server-side resource called a Photomap. The arrow shows how image data flows in the photoflo graph. Much more ambitious photoflos are possible. To summarize, the rules involving photoflo topologies are:

- All paths in a photoflo must start with an import element, may be followed by one or more process elements, and must end with an export element.
- No cycles are allowed in a photoflo graph.

Techniques

As stated earlier, ImportClientPhoto reads image data sent by a client. The client must specify to ImportClientPhoto the encoding of the image data to be sent, and it must do this at the time the photoflo is being constructed. This is done by specifying a technique constant as an argument to the function that is used to add ImportClientPhoto to the photoflo graph—the photoflo element convenience function. For example, to decode TIFF-PackBits-encoded image data, the client passes the constant **xieValDecodeTIFFPackBits** as an argument to **XieFloImportClientPhoto**. In addition, most techniques require a set of technique parameters. These define more precisely how the technique will carry out its task. Technique parameters are specified by passing a pointer to a structure containing the needed information. XIElib provides convenience functions that can be used to allocate and initialize these structures. Most import, process and export elements support techniques and technique parameters. In some cases, a default technique can be specified by the client. In this situation, technique parameters are not supplied, as the server decides upon appropriate defaults.

Parameters common to all techniques are supplied using arguments to the photoflo element convenience function. For example, XieFloImportClientPhoto takes width, height and levels arguments. The levels argument is an array of three long integers that specifies, per band, the depth of the image and how many distinct colors it can represent. If we are dealing, for example, with a 24-bit color image, levels would be set to {256,256,256}.

Import Elements

These read data from the client or from server resources. Import elements are how image data is made available to a photoflo for processing. The import elements supported by XIE are shown in the sidebar "Import Elements".

Process Elements

These read image data from elements earlier in a photoflo graph and manipulate the image data before passing it along to downstream elements.

Most process elements are able to handle both SingleBand (gray scale or two-tone) or TripleBand (color) image data transparently. Some process elements allow only one input, some operate on one or two inputs, and one (BandCombine) requires three SingleBand inputs. The output of a process element is image data. For example, Arithmetic takes two images, or an image and a constant, and adds them together pixel-by-pixel; the result is its output. Process elements supported by XIE are shown in the sidebar "Process Elements". Export elements supported by XIE are shown in the sidebar "Export Elements".

Handling Image Data

One thing lacking in XIE is client-side support for the handling of image file formats. An image file format defines how image data is stored on disk. For example, JPEG-encoded image data can be stored in TIFF or JFIF formatted files. Why are file formats needed? They organize the way image data, palettes and header information describing image width, height and depth are stored in a file. The problem with XIE is that clients need to find out what a particular file contains and provide the code needed to read image data and header information from the file. Once again, header information is needed because the client must tell XIE about the image data it will be sending to ImportClientPhoto.

The Internet contains good resources for dealing with this problem. See the XIElib example code section of my home page (the URL is given at the end of this article) to download example code dealing with this issue. The example code is based upon two libraries, supplied along with my examples, and available elsewhere on the Net. The first, libtiff, can be used to read header information and encoded image data from TIFF files. The other library can be used to extract header information and encoded image data from JFIF (JPEG) format files.

An Example: Employee Database

Let's turn to the specifics of adding image support to a fictitious application. Figure 3 illustrates a simple Motif client I will develop in the remainder of this article. The example client reads the current directory for files ending with .emp. Information about a specific employee is stored in these files. For example:

```
123
12
Barney Smith
1124
Boogie Woogie Avenue
Bedrock
CA
91911
Individual Contributor
32000
barney.img
```

The first line is the employee number (123). The second is the department number (12). The line immediately preceding the last is his salary (32000). The last line is a file containing a 256x256 JPEG image of the employee. For simplicity's sake, I used fixed width and height images to avoid the need to perform scaling. This is not because XIE doesn't support scaling. XID's **Geometry** element, which is used to perform scaling, would require its own article to describe fully.

Figure 3. Motif Client Displaying Employee Picture

In Figure 3, the employee numbers are displayed in a scrolled list on the left-hand side of the GUI. As the user clicks on an employee number, the GUI displays the employee's picture and other data read from the file.

In the following code, I'm going to ignore issues related to the reading of records and the Motif GUI. If you have specific questions about these areas, I'd be glad to answer them by e-mail. Think of our task as follows: we have a Motif application that does everything described above except it lacks the ability to display the employee's picture. We've added the Motif code needed to provide an area into which the image is to be displayed (using a **DrawingArea** widget). Our task is to add the image display feature to the application, and we are required to come up with a solution that uses XIE.

All XIE client applications must include the file XIElib.h using the following preprocessor directive:

```
#include <X11/extensions/XIElib.h>
```

The following include file is my own creation, and is supplied on my web site with the example code. It is needed to define data structures used by the photoflo backend code that I will describe later in the article.

```
#include "backend.h"
```

Before main, a couple of static globals are also declared:

```
static XieExtensionInfo *xieInfo;
static XiePhotospace photospace;
```

The first thing an XIE application must do is establish a connection to the XIE extension.

This is done after connecting to the display. In our main application, just after we establish the connection to the server using **XOpenDisplay** or some equivalent, we connect to XIE using the following code:

```
if ( !XieInitialize( display, &xieInfo ) ) {
fprintf( stderr,
        "XIE not supported on this display!\n");
exit( 1 );
}
```

The variable **xieInfo** is a handle that represents the connection to XIE. Its fields contain information about the capabilities of XIE on the server pointed to by **display**. Most clients need not concern themselves with the contents of the structure, except when dealing with XIE errors and events (which I won't discuss here).

Next, we declare a photospace. This represents a context in which immediate photoflos can execute on the server. Immediate photoflos are created and executed using a single XIElib API call named **XieExecuteImmediate**. After an immediate photoflo executes, it is destroyed by the X server. A stored photoflo, on the other hand, persists on the server until explicitly destroyed, or the creating client breaks the server connection and no other clients are referencing the photoflo. Stored photoflos can be executed more than once. Our example client needs a photospace, since we are executing immediate photoflos in our example. This is done by calling **XieCreatePhotospace**:

```
photospace = XieCreatePhotospace( display );
```

Before we call **XtAppMainLoop** to handle the GUI of the application, a call is made to a routine we provide named **LoadEmp**. LoadEmp reads all of the .emp files found in the current directory and stores them in a linked list of **EmpDat** structures. LoadEmp also calls a routine, **LoadImage**, passing a pointer to the **EmpDat** structure containing the name of the image data file. LoadImage reads the file and stores the image data on the server, using XIE. The image data read by LoadImage is stored in JFIF files and is encoded as JPEG Baseline, an

encoding supported by XIE. LoadImage supports both color and gray-scale JPEG images.

Let's take a close look at LoadImage. What LoadImage does is the following:

- It creates a Photomap resource on the X server. The Photomap resource will be used to hold or cache the image data on the server so that we transport it over the network to the X server only once. The image data will be stored in the Photomap resource in an uncompressed format.
- It reads the JPEG image from the specified file, obtaining the image data as well as the header information, i.e., width, height, levels and number of bands (gray scale or color).
- It constructs a photoflo to receive the image data from the client, decode it and store it in the Photomap resource. The photoflos used to handle color image data and gray-scale image data differ. Both photoflos use ImportClientPhoto to receive and decode the image data, and ExportPhotomap to store the result in the Photomap resource. If the image is TripleBand (color), we add a third element, **ConvertToRGB**, between ImportClientPhoto and ExportPhotomap to convert the image data from the YCbCr color space to the RGB color space. All image data displayed in a window must correspond to the RGB color space, since video displays are RGB devices.
- It executes the photoflo and sends the image data.
- It stores the resource ID of the Photomap resource in the **EmpDat** structure passed by reference to LoadImage so that we can make use of it later. We also record the number of levels in the JPEG image (3 or 1); this is needed to determine how to generate pixel data from the image data prior to display.

Here is the LoadImage code:

```
int
LoadImage( EmpDat *newp )
{
int floSize, size, decodeTech, floId = 1, idx;
Bool notify;
short w, h;
char d, l, *bytes;
XieConstant bias;
XiePointer decodeParms;
XiePhotoElement *flograph;
XieYCbCrToRGBParam *rgbParm = 0;
XieLTriplet width, height, levels;
```

Now we create a photomap resource and store the result in **newp** for later use.

```
if ( ( newp->pmap = XieCreatePhotomap( display ) )
        == (XiePhotomap) NULL ) return( 1 );
```

GetJFIFData is a routine available on my web site that reads JFIF files for image data and header information. We use it next:

```
if ( ( size = GetJFIFData( newp->image, &bytes,
        &d, &w, &h, &l )) == 0 ) {
XieDestroyPhotomap( display, newp->pmap );
fprintf( stderr,
        "Problem getting JPEG data from %s\n",
        newp->image );
return( 1 );
}
newp->bands = l;
```

This example only supports 8-bit gray-scale or 24-bit color (8,8,8) image data.

```
if ( d != 8 ) {
XieDestroyPhotomap( display, newp->pmap );
fprintf( stderr, "Image %s must be 256 levels\n",
        newp->image );
return( 1 );
}
```

XieAllocatePhotofloGraph allocates a photoflo graph which we then fill in with elements. If we are dealing with gray-scale image data (l == 1), we need only two elements. If we are dealing with color image data (l == 3), we need a third element to convert the image from YCbCr color space to RGB.

```
floSize = (l == 3 ? 3 : 2 );
flograph = XieAllocatePhotofloGraph(floSize );
```

Set up the width, height, and levels arguments to **XieFloImportClientPhoto**. This information was obtained by reading the header information from the JFIF file.

```
width[0] = width[1] = width[2] = w;
height[0] = height[1] = height[2] = h;
levels[0] = levels[1] = levels[2] = 256;
```

The image, SingleBand or TripleBand, is JPEG Baseline, so specify the corresponding decode technique and allocate the needed technique parameters. The decode technique and technique parameters are also passed to **XieFloImportClientPhoto**.

```
decodeTech = xieValDecodeJPEGBaseline;
decodeParms = ( char * ) XieTecDecodeJPEGBaseline(
        xieValBandByPixel, xieValLSFirst, True);
```

Now we can add ImportClientPhoto as the first element of the photoflo graph.

```
idx = 0;
notify = False;
XieFloImportClientPhoto(
&flograph[idx],  /* address of element
                  * in photoflo graph */
(l == 3 ? xieValTripleBand : xieValSingleBand),
                /* data class */
width,       /* width of each band */
height,      /* height of each band */
levels,      /* levels of each band */
notify,      /* send DecodeNotify event? */
decodeTech,  /* decode technique */
decodeParms  /* decode parameters */
);
idx++;
```

If the image is color, then convert from YCbCr to RGB. **XieTecYCbCrToRGB** is used to allocate the technique parameter needed by the **YCbCrToRGB** technique. Both the allocated technique parameters and the technique are

passed to **XieFloConvertToRGB**, which is used to add the ConvertToRGB element to the photoflo graph. It is beyond the scope of this article to discuss the arguments and technique parameters used, but the code below should work for most color JPEG Baseline images encountered by an application.

```
if ( l == 3 ) {
bias[ 0 ] = 0.0;
bias[ 1 ] = bias[ 2 ] = 127.0;
levels[ 0 ] = levels[ 1 ] = levels[ 2 ] = 256;
rgbParm = XieTecYCbCrToRGB( levels,
        (double) 0.2125, (double) 0.7154,
        (double) 0.0721, bias, xieValGamutNone,
        NULL
        );
XieFloConvertToRGB( &flograph[idx], idx,
        xieValYCbCrToRGB, (XiePointer) rgbParm
        );
idx++;
}
```

The final element in the photoflo is **ExportPhotomap**. The encode technique used is **xieValEncodeServerChoice**. Given the photoflo we are dealing with, this should cause XIE to store the image in an uncompressed, canonical format within the Photomap resource.

```
XieFloExportPhotomap( &flograph[idx], idx,
        newp->pmap, xieValEncodeServerChoice,
        (XiePointer) NULL);
idx++;
```

Now that we have a photoflo graph, we can send it to the server and start its execution by calling **XieExecuteImmediate**:

```
XieExecuteImmediate( display, photospace, floId,
        False, flograph, floSize );
```

Once execution starts, the photoflo will be blocked, awaiting image data from the client. The XIElib function that sends this data is **XiePutClientData**, and it can be used to send any client data (ROIs, LUTs and images) to the ImportClient element awaiting the data. **PumpTheClientData** is a utility function I wrote (also available on my web site) that is a wrapper around XiePutClientData and makes the process of sending data to an **ImportClient** element a little easier.

```
PumpTheClientData( display, floId, photospace, 1,
        bytes, size, sizeof(char), 0, True );
```

At this point, the image data has been read by the photoflo, decoded, converted to RGB color space (if it was color) and stored in the server-side Photomap cache for later use. In addition, the photoflo we executed has been destroyed by the server. Now, we need to free the memory allocated to the photoflo graph and other items in the above code.

```
if ( rgbParm )
XFree( rgbParm );
free( bytes );
XieFreePhotofloGraph( flograph, floSize );
XFree( decodeParms );
return( 0 );
}
```

We need code that will transfer the image data from a Photomap resource to a window. Two different situations will cause the client to perform the actual drawing:

1. The user selects an employee number from the scrolled list.
2. The DrawingArea widget's window becomes exposed, and server backing store is not enabled.

Let's write the code to handle the first case. After we created the scrolled list widget, we registered with the widget a callback to be invoked whenever the user selects an item in the list:

```
XtSetArg(args[0], XmNselectionPolicy,
        XmSINGLE_SELECT);
list_w = XmCreateScrolledList(rowcol,
        "scrolled_list", args, 1);
XtAddCallback(list_w, XmNsingleSelectionCallback,
        ListCallback, NULL);
XtManageChild(list_w);
```

Thus, when the user clicks on an item, Xt will call the function **ListCallback**. Inside of ListCallback, we perform the following tasks:

- Determine which employee was selected, and obtain a pointer to the corresponding **EmpDat** structure.
- Update the text fields in the dialog with information about the employee (e.g., name, department, address and salary).
- Call a function to display the employee image in the window owned by the DrawingArea widget.

The following is my implementation of ListCallback:

```
static void
ListCallback(Widget list_w, XtPointer client_data,
        XmListCallbackStruct *cbs)
{
char    *choice, buf[ 32 ];
EmpDat *p;
/* Read the list item, and then look it up in our
 * linked list of employee records */
XmStringGetLtoR(cbs->item, charset, &choice);
p = FindChoice( choice );
XtFree(choice);
/* If we have a match, display the text
 * information in the dialog */
if ( p != (EmpDat *) NULL ) {
/* first do the text fields */
sprintf( buf, "%d", p->code );
XmTextFieldSetString( codeT, buf );
XmTextFieldSetString( nameT, p->name );
XmTextFieldSetString( streetT, p->street );
XmTextFieldSetString( cityT, p->city );
XmTextFieldSetString( stateT, p->state );
XmTextFieldSetString( zipT, p->zip );
XmTextFieldSetString( descT, p->desc );
sprintf( buf, "%ld", p->salary );
XmTextFieldSetString( salaryT, buf );
/* Go and display the image. gDrawP is discussed
 * later */
gDrawP = p;
DisplayPhotomap( p );
```

```
    }
    }
```

The routine that does the real work associated with transferring the image data from the photomap to a window is **DisplayPhotomap**. It is a separate routine (i.e., not part of ListCallback), because we need to call it when handling window exposures.

```
    void
    DisplayPhotomap( EmpDat *p )
    {
    XiePhotoElement *flograph;
    Visual *visual;
    Backend *backend;
    int floId = 1, screen, idx, floSize, beSize;
    Display *display;
    if ( p == (EmpDat *) NULL ) return;
```

The first thing we do is generate a backend for the photoflo we are constructing. A backend is a set of process elements, plus **ExportDrawable** or, if the image is two-toned, **ExportDrawablePlane**. The purpose of these elements is to prepare the image data for display in the specified window. The backend is responsible for the following:

- Ensuring that the levels attribute of the image corresponds to the target window. For example, if the image is 8-bit color, and we are displaying to a 1-bit StaticGray window, we must insert a Dither element into the backend to reduce the levels of the image from 256 to 2. Dither is the best way to preserve the contents of the image in these cases. Other process elements can do the job but will mangle the result quite a bit. If the levels attribute of the image and the capabilities of the window do not match, XIE will generate an error and abort the photoflo.
- Converting image intensity values to color-map index data. From Xlib programming, recall code such as the following which allocates the color red from a specific color map:

```
            Display *display;       /* server connection */
            int screen;     /* usually 0 */
            Colormap cmap;  /* resource ID of color map */
            XColor color;   /* holds info about a color */
            cmap = DefaultColormap( display, screen );
            color.red = 65535;
            color.green = color.blue = 0;
            XAllocColor( display, cmap, &color );
```

  Now, we can use the returned color to draw, for example, a red line in a window by setting the foreground color of the GC we associate with the window to the pixel value returned by **XAllocColor**:

```
            XSetForeground( display, GC, color.pixel );
            XDrawLine( display, window, gc, x1, y1, x2,
                    y2 );
```

  Thus, when we want to draw a line of a particular color in a window, we actually draw to the window the pixel value which indexes the color in the color map associated with the window. The same thing has to happen

when displaying images. X expects our window to contain pixel values. The server (hardware) takes these pixel values and converts them to colors that we see as the screen is refreshed. A convenient way to map colors in our image to a set of pixel values is to add a **ConvertToIndex** element to the photoflo backend. ConvertToIndex's job is to translate all of the color values into pixels and allocate any cells needed in the color map.

- The final task of the backend is displaying the image in the window. If the image is gray scale or color, we add an ExportDrawable element as the last element of the backend. If the image is two-toned, then we use ExportDrawablePlane.

In order to generate the backend, I make use of routines available on my web site. **InitBackend** takes information about the image and target window and determines which elements are needed to complete the backend processing. It also returns the number of elements needed, so that we can allocate space for them in the photoflo graph.

```
display = XtDisplay( drawingArea );
screen = DefaultScreen( display );
visual = DefaultVisual( display, screen );
if ( p->bands == 1 )
        backend = (Backend *)
        InitBackend( display, screen,
        visual->class, xieValSingleBand,
        1<<DefaultDepth( display, screen ),
        -1, &beSize );
else
        backend = (Backend *) InitBackend( display,
        screen, visual->class,
        xieValTripleBand,
        0, -1, &beSize);
        if ( backend == (Backend *) NULL ) {
                fprintf( stderr,
                "Unable to create backend\n" );
        exit( 1 );
 }
```

Now that we have taken care of the backend, we allocate the photoflo graph and add ImportPhotomap as its first element. We pass to XieFloImportPhotomap the resource ID of the photomap from which the image should be read. This resource ID is stored in the **EmpDat** structure passed into this routine as an argument.

```
floSize = 1 + beSize;
flograph = XieAllocatePhotofloGraph( floSize );
idx = 0;
XieFloImportPhotomap( &flograph[idx],
        p->pmap, False );
idx++;
```

Next, a call is made to **InsertBackend**, which adds the backend elements to the photoflo graph.

```
    if ( !InsertBackend( backend, display,
            XtWindow( drawingArea ), 0, 0, gc,
            flograph, idx ) ) {
    fprintf( stderr, "Unable to add backend\n" );
    exit( 1 );
    }
```

Now that we have a photoflo graph, we call XieExecuteImmediate, which is responsible for transmitting the photoflo to the server and executing it. Since the photoflo is immediate, it will be destroyed by the server once execution completes. At this point, the image data in the photomap should be visible to the user in the DrawingArea widget's window.

```
    XieExecuteImmediate( display, photospace, floId,
            False, flograph, floSize );
    XieFreePhotofloGraph( flograph, floSize );
    CloseBackend( backend, display );
    }
```

The final routine to discuss is **RedrawPicture**. This simple routine is a callback, registered with the DrawingArea widget instance, to be called whenever the DrawingArea widget's window receives an **expose** event. Recall that ListCallback stored the pointer to the **EmpDat** structure corresponding the user's list selection to a global variable named **gDrawP**. Thus, **gDrawP** holds a pointer to the currently displayed employee data. All we need to do in RedrawPicture is check whether **gDrawP** points to valid data; if so, we know the user had previously made a selection. Now, we can call DisplayPhotomap, passing **gDrawP** as an argument, to render the image to the window.

```
    static void
    RedrawPicture(Widget w, XtPointer
    client_data, XmDrawingAreaCallbackStruct *cbs)
    {
    if ( gDrawP != (EmpDat *) NULL ) DisplayPhotomap(
            gDrawP );
    }
```

The Complete Example Code

The complete source code for the example discussed in this article, and the library routines needed to build it, can be found at my home page located on the Internet at http://www.users.cts.com/crash/s/slogan/. This article describes just one of over 40 example clients you will find there. My book on XIElib programming, *Developing Imaging Applications with XIElib*, published by Prentice Hall, goes into much greater detail than I could provide in an article of this length. More information on my book can be found on my web site as well or at http://www.prenhall.com/. If you have any questions about XIE, this article, my other examples, or for that matter X11, please feel free to drop me an e-mail.

Depending upon the phase of the moon, you'll find Syd developing software for Macintosh (Apple's MacX 1.5 and 2.0), the X Window System (Z-Mail for UNIX) and even Windows NT (NetManage's NFS client). He was a member of the team that produced the XIE example implementation for X11R6. In his spare time he enjoys buzzing around the San Diego coastline in Cessnas and Piper Archer IIs. He can be reached at slogan@cts.com.

Archive Index Issue Table of Contents

Advanced search

Open Inventor

Robert Hartley

Issue #53, September 1998

Mr. Hartley shows how to do interactive 3-D programming using Open Inventor, Release 2, which he used to create the images on our cover.

Open Inventor is a powerful 3-D graphics library that allows the user to create interactive 3-D applications quickly and easily. It adds object-oriented programming to OpenGL, the most widely available standard 3-D API. This layer of object-oriented abstraction does not come at the expense of losing control of our applications—all the power of programming directly with OpenGL is still available.

One measure of how much detail is needed in order to get things done is the size of the standard reference books. My personal top five essential books on graphics programming are listed in Resources. To be functional in Inventor programming, the most essential one is Josie Wernecke's *The Inventor Mentor*.

What is in it?

Inventor organizes its data into a scene graph, a structured collection of graphical objects stored as nodes. These nodes can represent many things, from geometric primitives, engines, lights and material properties to transformation nodes that can include scaling, rotation and translation properties.

Inventor efficiently handles many of the graphic operations which would otherwise have to be coded by the user. It has facilities for scene graph management, picking, viewing and user interaction. The standard viewers come in five basic flavors: fly, walk, plane, examiner and render area.

Editors for materials, directional lights, transformations and other node properties can be attached to the scene graph, and changes rendered

interactively in any of the standard editor viewers. These editors come in source form, so they can be customized to suit the user's specific needs.

Where did it come from?

Open Inventor was developed by Silicon Graphics (SGI), a company that builds graphical workstations. It is the second version of Iris Inventor, which encapsulated IrisGL, from which OpenGL is derived. The Inventor/GL API is richly featured and has been maturing for about a decade now. It has proven worthy of the time and effort needed to learn it. Although there is a lot to learn, much can easily be done even by a newbie, as will be demonstrated shortly.

Template Graphics Software (TGS) has a source license to Open Inventor and OpenGL. TGS takes care of porting and distributing Inventor to Linux and other UNIX platforms and Windows NT/95, enhancing and enriching it along the way. In September 1997, the first version of Inventor for Linux appeared on the TGS Internet site (see Resources).

What is the VRML connection?

The Inventor file format was chosen as the basis for the VRML version 1.0 file format, commonly distinguished by its .WRL extension. Many of the Inventor nodes were used directly in VRML, and it has often been referred to as “Inventor with all of the good stuff ripped out”. With the new definition of VRML 2.0, the recent releases of Inventor, including the latest Linux version, have been updated to be able to read, write and process VRML files. Actually, I have found the standard Inventor viewers make better VRML model viewers than some of the ones available for Windows and UNIX/Linux. They provide better performance and a better rendering appearance.

What can I do with it?

Inventor is an ideal environment for creating animations, simulations, data visualizations, VR work and CAD.

At Pratt & Whitney, we use Catia on high-end UNIX workstations for our design/manufacturing process in the development of gas turbine engines for aircraft, industrial and marine applications. Catia is a very powerful 3-D CAD/CAM system, and has its own API for querying drawing models to retrieve geometry and other information.

Using this API, an Open Inventor program **xmtriag** was written to convert Catia solid models to stereolithography (STL) format, ready for input into an STL machine for rapid prototyping. In addition to the STL format, the program

generates an Inventor file for us to visually verify that the part was translated correctly.

My company uses this program as a utility in our design process to take a 3-D snapshot of an engine part, convert it to STL format and send it via FTP to a supplier, who then sends us back a quote and, eventually, a part. These parts can be used for design reviews or for casting purposes to make precision molds. This method saves both money and time in the design/review/ manufacture cycle and increases quality. In fact, this process has been so successful we are in the process of installing our own in-house rapid prototyping facility.

Besides those in engineering, many others are interested in seeing and querying our drawing database. To avoid tying up Catia licenses, a viewer was developed to extract the 3-D geometry from the model database and convert it to Inventor format. These new Inventor models can be kept as a light version of the parts instead of storing them in Catia. Using an SQL database, we can select parts from a hierarchical drawing tree regardless of the platform being used and view associated engineering and other data of the selected parts. In addition to Catia, this converter/viewer will process Unigraphics and other CAD drawing formats.

As a result, we have not had to buy new Catia licenses or other expensive third-party viewers. The bulk of the graphics development was done by one person, with some ancillary help from database-oriented personnel over a period of six months. Excluding the price of the Inventor licenses, which are free on SGI workstations, the ratio of prices for third-party viewer licenses to this person's salary was over 100 to 1.

How does it work?

Inventor contains mostly 3-D objects and their associated attributes: geometric shapes, colors, lights and 3-D object manipulators. These are rendered with OpenGL or a similar API, such as Brian Paul's Mesa (see Resources).

The Inventor Xt Component and Utility Library provides widgets and utilities for event handling, rendering, viewers and editors that can manipulate the scene graph directly. These events include selection, picking and highlighting of nodes, keyboard and mouse handling, and processing Xt and Motif callback functions.

What else do I need?

For standard GUI design, we can resort to Motif (MFC under Windows NT/95), or whatever happens to be our favorite 2-D GUI API. The standard components

such as the predefined viewer classes have been set up to make use of Motif and Xt under UNIX/Linux using the SoXt classes. These classes have been emulated under Windows, so that we have a somewhat easier time of porting code between platforms.

I find Motif to be a little cumbersome to program, so I've been using Viewkit, available from http://www.ics.com/, in conjunction with RapidApp running on Irix to build the GUI and generate the C++ code stubs. Viewkit is freely available for SGI and Linux workstations, and the RapidApp code compiles cleanly under Linux. The GUI could have been built by hand, totally under Linux, but it helps the design process to be able to work interactively. Also, building on both platforms may help to make sure that portability issues spring up earlier in the design process.

Motif is not an absolute prerequisite under UNIX. Mark J. Kilgard, of SGI until recently, illustrates using Open Inventor without Motif in his book *Open GL Programming for the X Window System* (see Resources).

"Hello Cone", an Inventor Sample

Let's look at a quick illustration of how easy it is to set up a scene graph and viewer. Example 2-4 of *The Inventor Mentor* presents "Hello Cone" using a standard scene viewer. (See Figure 1.) The figure shows three main areas of the SceneViewer: three thumbwheels, eight side buttons and a render area.

Figure 1. Scene Viewer with "Hello Cone"

The eight buttons handle object selection/picking, viewpoint manipulation, help, returning to home viewpoint, setting a new home viewpoint, executing **viewAll** to see the whole scene, seeking to a point and toggling the camera type between orthographic and perspective.

The three thumbwheels handle manipulation of the scene's viewpoint by rotating the camera angle about the X and Y axes and traversing along the Z axis to obtain a zoom effect.

The render area is the most interesting. The mouse can be used to get the same effect as the thumbwheels. Mouse button one will allow the user to select the object, and if the mouse is moving when it is released, the object will continue spinning in the direction that the mouse cursor was moving. Mouse button two will allow the user to pan up/down and left/right, and if the control key is pressed, zoom in and out. Mouse button three causes a pop-up menu to appear that allows the user to set various attributes (such as rendering "as-is",

hidden-line, wire frame, points-only and bounding box), preferences and displays of the thumbwheels and side buttons (known as decorations).

Listing 1 shows how simple it is to create this program. The first seven lines are the minimum header files. Inventor has 553 different include files. This may seem like a lot; however, each is very specialized, and selecting only the needed ones will speed up compilation time. If I wanted to, I could have simply included Inventor/So.h and let the compiler process all of the “So” prefixed files.

The first two executing lines after **main** create the main window widget and invoke **SoXt::init**. This is an essential part of the program, because here, Inventor is bound to Xt event handling so that its sensors will work properly. SoXT::init is also where the licensing code is called. Failing to invoke **init** will result in a core dump.

To be visible, each scene graph must have a node to attach to a viewer. In the listing, I am using a **SoSeparator** which saves the traversal state before being entered and restores it afterwards. This serves to prevent the attributes of its child nodes from affecting other parts of the scene graph that follow. A separator can include lights, cameras, coordinates, normals, bindings and all other properties. Separators also provide caching and culling of their children based on bounding box calculations during picking and rendering.

Once you create a node and pass it to the scene graph, Inventor takes over. Inventor nodes are always created dynamically with the C++ **new** command—never on the stack. Each node has a reference count, starting with zero when created, and incrementing and decrementing as nodes are added and removed as children to other parent nodes. When this reference count drops from one to zero, Inventor automatically deletes the node.

During traversal, the node is referenced and then dereferenced as the scene graph is traveled over. If we had not done a **ref**, the first time we traversed the scene graph its reference count would have incremented as it was entered, moving its reference count to one, and then decremented it back to zero as it was left, automatically deleting the node and any children whose reference counts had also dropped to zero. We would have been left wondering where our node(s) went.

Next, we add a material property as a child to the root node. It has a diffuse color or (1.0,0.0,0.0), which corresponds to full red, with red, green and blue (RGB) quantities being expressed as floating-point values between 0.0 and 1.0.

A cone is now added to the root node. Its default values are one unit for base and one for the length. It is located at the origin 0,0,0, and when unrotated,

points one unit up from the base along the Y axis. Since the cone comes after the material property specifying the color red mentioned above, the cone inherits its attributes and is also red.

Inventor traverses its scene graph by starting at the root node and traveling down and to the right. Since OpenGL is a state machine, once we set an attribute, it will retain that value until changed.

Now we create the SceneViewer, passing the widget of our parent window, hooking our scene graph to it and setting the window title.

The **show** and **hide** methods call **XtManageChild** and **XtUnmanageChild** if a sub-widget is passed to it. If the widget is a shell widget, show will call **RealizeWidget** and **XMapWindow**, and if it is iconised, it will raise and de-iconify it. The hide method will call **XUnmapWindow**.

Inventor Manipulators

This program can be enhanced to allow more than the viewpoint to change by adding a manipulator. Using **SoTrackballManip** provides the ability to interactively rotate an object about its own center on any of the three axes, as shown in Figure 2.

Figure 2. Rotating the Cone

Only two lines need to be added to the source code. At the end of the includes section of the program above, add the line:

```
#include <Inventor/manips/SoTrackballManip.h>
```

and immediately after the **root->ref** call, add the line:

```
root->addChild(new SoTrackballManip);
```

To be sure I was making the cone rotate, I moved it off to the side using mouse button two, and turned on the origin's three axes display using the pop-up menu of mouse button three.

Clicking the top button on the right of the main window changes the mouse to selection mode. Moving the cursor to any intersection of the double rings and selecting it with mouse button one causes it to be highlighted in yellow, allowing us to rotate the cone about its axis. If the mouse is still moving when released, it will continue spinning. Clicking on the hand icon will revert to viewpoint manipulation mode, so that both the cone and its manipulator can be rotated about the viewpoint axis. The cone is now spinning on its own axis and orbiting around the center of the scene.

Other manipulators will perform rotations, scaling and translation (movement) tasks and any combination of these functions.

Editing Nodes

Inventor comes with a number of standard node editors for modifying the scene. Figure 3 is a typical model (without any alterations) in SceneViewer, a demonstration program that comes with the Inventor distribution. For a little more information about the figure, see the sidebar "A Bit About Figure 3".

Figure 3. A model of Deep Flight One, the single person submarine designed and built by Hawkes Ocean Technologies of San Raphael, California

Selecting Editors from the main menu, we can select "Material Editor" and get the dialog shown in Figure 4.

Figure 4. Material Editor Dialog

The blue dome can be selected on the front of the craft and made semi-transparent; a few lights can be added and the viewer's built-in head lamp turned off. Inventor has three types of lights which can be added to a scene:

- **SoPointLight** acts like a light bulb, radiating light equally in all directions.
- **SoDirectionalLight** emits light along a single direction as though it were nearly an infinite distance away, like the sun.
- **SoSpotLight** emits light in a cone, allowing it to be pointed and focused.

Figure 5 is an example of a green SoDirectionalLight and red and white SoSpotLights. Their respective icons have been left on so the manipulators are visible—they would normally be left off during rendering. With the mouse, I can select and move them as well as change their direction. The SoSpotLights have the added cone to allow focusing the light by changing the radius of projection.

Figure 5. Results of Intersecting Color Spotlights

In Figure 5, we see how light behaves when pools of light intersect. Having changed the dome's material properties, we see how the green light appears as a reflection. If fog, other atmospheric effects and texture mapping were added, we would have a highly realistic scene, ready for imaginative animation.

This submarine model is part of a simulator on which I am working. It includes a digital elevation map viewer that reads and renders topographical data to display terrain in which to drive. I used some of John Beale's ideas about making use of government data in rendering landscape images. On his site at http://www.best.com/~beale/, he provides a few useful converters, which he

has adapted from others, to turn the one minute elevation maps into PGM file pixmaps.

Figure 6. Terrain Display

The terrain-handling section of the program is shown in Figure 6. It allows control of terrain resolution and color. The viewer on the right side is standard, so the image can be zoomed, panned and rotated as desired. Depending on machine speed and terrain complexity, the rendering is more or less interactive. Personally, I have found a value of 24 to be an ideal terrain-complexity value.

The elevation data is kept in a 1204 by 1204 integer array representing the heights surveyed in the map. This would be a bit much to try manipulating interactively on a PC, so the terrain-complexity thumbwheel allows us to choose the amount of reduction we want. A very simple algorithm is used. The value of the thumbwheel, two in Figure 6, is used as the length of a square from which we take the maximum height, and then added as a coordinate point to an Inventor **SoQuadMesh**, basically a grid composed of quadrilaterals. With a value of two, the maximum height is four elevation points. A value of ten provides a maximum height of 100 points, eliminating 99% of the map for rendering purposes. As this works interactively, the value can be changed until a balance is found that combines the amount of needed detail with the desired rendering speed. Within this, the area of interest can be located and only the needed detail rendered.

Each SoQuadMesh node in Figure 6 is defined by a set of four vertices, each shared with its adjacent neighbors. These vertices have an associated color component which is smoothed out over the area of the SoQuadMesh. To give some visual indication of the height and to generally make things more interesting, I have set each elevation to have a different color based on a simple formula specified by the user.

The six vertical wheels handle the assignment of maximum and minimum depths for each color component, so overlapping blue and red results in a purple area of overlap. The amount of color is calculated as follows: from the maximum and minimum ranges the midpoint is found, and the percentage of distance from the color border to this midpoint is the percentage of that color to use. Thus, a vertex at the midpoint has a value of 1.0 for that color component and a vertex three quarters of the way to the edge will have a value of 0.75 of the color.

The file format read in is basically an ASCII PGM file containing three values representing the maximum X, Y and Z values, followed by all elevations. This is

handy because the file can be read with the **xv** command and saved in any other graphical file format needed.

Originally, the terrain represented in Figure 6 was the pixmap shown in Figure 7, with lighter areas having more height.

Figure 7. Original Pixmap Terrain Image

Hooking Up New Hardware

Joystick drivers are available for Linux. *The Inventor Toolmaker* book on how to extend Inventor has a section in the back on adding new hardware devices. A lengthy discussion ensues, requiring a knowledge of the details of the windowing system. I found I could bypass it by using an Inventor **SoTimerSensor** node set to invoke the callback function at almost any time interval with the attached node pointer. The setup for it is as follows:

```
SoTimerSensor *JS_Sensor =
 new SoTimerSensor(JS_SensorCallback, craft_xf);
JS_Sensor->setInterval(0.005); //scheduled 200/sec ...
JS_Sensor->schedule();
```

The function, called **JS_SensorCallback**, checks the joystick driver to see how much it has moved. This is also the place to update the flight state model to change the velocity and heading. We can check for things such as amounts of air and power left, and how deep we are, based on the current XYZ coordinates.

Things to Add

A few things I wish to add to this simulator are:

- Networking over the Internet to join two users together
- Animated objects such as fish
- Direct reading of the terrain data from the Internet
- More empirical parameters to ensure the craft responds the same way in the simulator as in life
- Tidal and water flow effects
- Light attenuation as depth increases

Summary

In addition to ease of use and availability of raw power, I like Open Inventor because whatever I develop is totally portable between my Linux machine at home and my SGI workstation at the office. Using Viewkit, Motif and Mesa, I have a tremendous amount of flexibility in choosing where and how I develop software.

Inventor and Viewkit provide the software functionality of an SGI workstation, and with the recent advances in graphics hardware, Linux PCs will be closer tomorrow to where the SGI workstations are today. Tomorrow is already planned—no one has to ask us where we want to go today.

A special thanks goes out to Alexandre Naaman for showing the way, being patient and simplifying things when they got muddled.

A Free Alternative to Inventor

Hardware and Software

Resources

Robert Hartley is an Australian citizen currently residing in Montréal, Canada. His first involvement with UNIX was on the Minix operating system in the mid-eighties. He started off his working life writing public security software for various municipalities around Quebec and Ontario. He is now a graphics systems analyst at Pratt & Whitney Canada. He can be reached via e-mail at robert.hartley@pwc.ca.

Archive Index Issue Table of Contents

Advanced search

LibGGI: Yet Another Graphics API

Andreas Beck

Issue #53, September 1998

The next generation fully portable graphics library

We didn't like the idea of another graphics library, but when we checked the available solutions at the time the GGI project was initiated, we found nothing that would fit all our needs:

- Portability
- Simplicity
- Transparent acceleration support
- Multihead support
- Extensibility
- Small size

Most of these issues had been addressed by one interface or the other, but there was none that fulfilled them simultaneously. Let us talk about those issues in more detail.

Portability

First of all, portability is our only weapon against the commercial software market. If we are so portable that we can run on any platform, including the mainstream market, we might be able to get those nice programs, because it is no problem to port them.

The X Window System is about as portable as a program gets and X applications are normally fairly portable. However, using X is often overkill and causes considerable overhead. Moreover, writing X programs is rather difficult (depending on which toolkit you use) and seems really alien to most non-UNIX gurus.

However, X is the most important platform in the UNIX world and, to call ourselves portable, we need to support X. LibGGI uses a system of dynamically loadable target drivers that allow it to run on anything with the ability to display graphics. It does not make a difference if the display target is some type of server software, a KGI-like device, something directly accessing graphics hardware, a printer, a system-service of a microkernel OS or something else. Table 1 shows a few available target systems that LibGGI programs can run on.

Table 1. LibGGI Platforms

Graphics-server systems:

- X Window System: AIX, IRIX, Solaris, Linux/x86/Alpha
- Microsoft Windows (very alpha)

Device-oriented systems:

- KGI: Linux/x86; planned: Solaris/x86, Linux/Alpha
- Solaris /dev/fb

Direct-access systems:

- VESA/DOS
- SVGAlib, GLIDE, SUID-KGI : Linux

LibGGI detects the most desirable target available on the current hardware and automatically makes use of it. This can be overridden to force different behaviour easily.

Compatibility is maintained at binary level within one platform. That is, a LibGGI application compiled for Linux x86 will run without modification on a KGI full screen, in an X window, using SVGAlib or GLIDE. It will even run on a text-mode screen via LibAA or whatever is available.

Compatibility across platforms requires a recompile, but this should be painless if the surrounding code doesn't heavily use OS specifics.

So, porting applications is easy. But what about porting LibGGI itself? We have tried keep LibGGI as portable as possible. Most GGI code compiles without a warning in **gcc -pedantic** mode. We have also tried to keep use of OS specifics to a bare minimum.

LibGGI should build easily on any system that has heard of POSIX. Even libdl isn't strictly required anymore to allow for systems that don't recognize dynamic libraries.

Simplicity: An Example

Another important point in the design of LibGGI is simplicity. If a programmer just wants to write a small graphics utility, he may be scared off by the complexity of X. To give you an idea of how programming with LibGGI works, let's look at the small example program shown in Listing 1.

It doesn't show good style, but is designed to be straightforward to read. As with any library, you have to include its headers. These are located in a subdirectory. Since we have more than one library, we decided that allocating a whole subdirectory would cause less confusion.

The first thing you have to do when using LibGGI is call **ggiInit**. This initializes the LibGGI internal data structures and sets up everything. Next, you call **ggiOpen**. This call returns a **ggi_visual_t** which is an opaque type, similar to what X calls a “drawonable”. Think of it as an abstract handle to the display you draw on. Note that you can have multiple displays per program as required by complex applications which handle multiple screens.

You will want to set a graphics mode on the visual. A mode consists of the size, or rather, resolution of the visible area (**visx**, **visy**) as well as that of the virtual area (**virtx**, **virty**) on which the view port can be panned around. Moreover, you need to specify the type of display you request; for example, a **GT_24BIT** true color visual. Note that calls to request additional options are available, as well as the capability to automatically choose values. This is highly recommended in order to enhance portability.

Graphics Context

Now we are set to start drawing. LibGGI uses a GC (graphics context) to represent the current state of the drawing system. We considered a state-free approach, but this would have meant:

- Lots of parameters for some functions
- A very awkward look for programmers used to the GC concept
- Ignoring that actual acceleration hardware normally has a GC

We now draw a few dots in different colors by using **ggiSetGCForeground** and **ggiDrawPixel**. As an alternative, we draw the next set of pixels using **ggiPutPixel**. Higher-level functions are also available, but only to a limited extent. As you can see from the example program, we support various kinds of lines and boxes (and yes, these are accelerated, if the underlying target supports it), but that's about it.

Don't be disappointed here. There is a higher-level library called LibGGI2D providing more complicated functions. LibGGI has been designed to be a basic “foundation” library on top of which specialized libraries can be built for more complex requirements, such as 3-D and animation.

Events

When we are done drawing, we use **ggiEventPoll** to wait for a key or mouse event. **ggiEventPoll** determines if an event of the given type(s) is present and will eventually block for it for a specified time or indefinitely, if the pointer to the **timeval** struct is **NULL** as in our simple case.

We then use a convenience function to get a keystroke. Note that this will block again, if polling was terminated due to mouse activity. In most cases, you will want to use LibGGI's event system to get input from any device that is attachable to a computer system. For event classification and configuration, a helper library called LibGII is available to give you a flexible and simple way of mapping device input to program actions. After **ggiGetc** has returned, we close down the visual using **ggiClose**, and then the whole LibGGI library using **ggiExit**. Note that you can reopen another visual before ggiExit, which can, for example, be used to transfer the program from one target to another. After ggiExit, every other call to LibGGI functions is undefined. You will need to call ggiInit again first. You have now gained a tiny glimpse at how LibGGI programs look.

Advanced LibGGI Usage

Many applications, especially those ported from DOS and other systems where a relatively direct access path to the hardware is present, will want to access graphics RAM directly. While being tied to the layout of the particular card/ mode isn't a great idea for portability, it is a good way to get extra speed. LibGGI solves this dilemma by exporting a **DirectBuffer** structure describing all details of the currently active video buffer. The application can decide whether to use it or fall back to standard LibGGI calls.

LibGGI applications can service multiple visuals at the same time, thus allowing multihead applications like CAD or games screens split over several monitors. For convenience, we have “memory-visuals” that can be used to draw an “invisible” area first and then blit to screen (crossblitting). Simple color-space management, such as gamma setting, is available, as well as support for double/triplebuffering and waiting for vertical retrace, or even for a specific position of the CRT beam (where the hardware allows).

3-D, Movies, Fonts

LibGGI ends at about the level of a DrawBox, which is not a desirable environment for many applications, and transparent acceleration is limited. LibGGI was kept small on purpose to work well under constrained conditions such as embedded systems, and not waste space for applications which do not need advanced functionality.

We extended LibGGI so more complex APIs could be implemented “on top” of it. So far we have LibGGI2D, Mesa-GGI and a tiny windowing library, LibGWT, running. A lightweight 3-D library, font and animation support are works in progress. Such libraries are implemented as LibGGI-Extensions. Being an extension has several benefits over just “using” LibGGI; for one thing, you inherit the complete functionality regarding library loading and target support. Thus, extension libraries also bring along their set of API drivers which can be used to allow for transparent acceleration. LibGGI ensures basic services, so all extension libraries will run on all LibGGI targets, but the level of acceleration will vary depending on the availability of driver libraries for the extension.

Transparent Acceleration and Multi-API

At the core of LibGGI is a trick that helps LibGGI be portable and smart regarding acceleration—a creative usage of dynamically loaded libraries. LibGGI functions can be overridden by loading a suitable library. LibGGI is aware of two different types of such libraries:

1. Display-Modules describe a way to connect to a given kind of back end like X, KGI, SVGAlib etc. They are loaded at ggiOpen time.
2. Driver modules are normally loaded at mode-setup time and each describes a given API used to draw on the current target. These APIs are normally selected by the back end that is queried for a set of “suggest-strings” that map to these APIs. See Figure 1.

Table 2. Multiple APIs

1. generic stubs (fallback emulation)
2. linear-8-framebuffer
3. generic-KGI-ioctl
4. vendor-ABC-KGI-PrivateIoctl
5. vendor-ABC-MMIO-DirectAccess

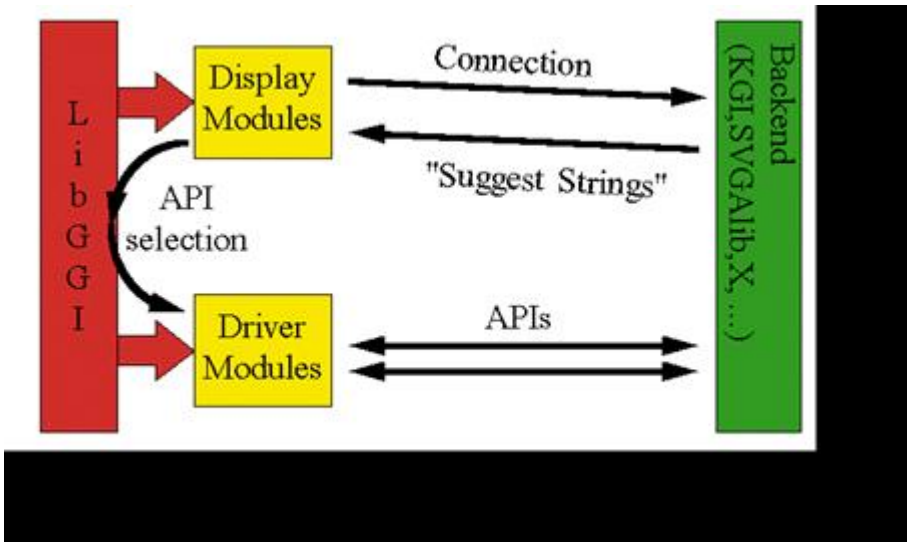


Figure 1. How Transparent Acceleration and Multi-API Work

You might be surprised by the term “set of”. Normally, there are multiple APIs which can be used to draw on a given target. Let me explain this point a bit further for the KGI target, which makes the most extensive use of this feature. Table 2 is the set of suggest-strings for a fictional ABC graphics card being accessed via KGI. The KGI module managing the card will tell LibGGI to first load a “stubs” library that is used for emulation when a function is not natively supported. This stubs library contains fall-back functionality, such as making up a filled box from multiple horizontal lines. Then LibGGI loads a library that accesses the linear, 8-bit wide frame buffer exported by KGI. This library will hold primitives such as **DrawPixel** and override the stubs library. LibGGI will then load KGI's generic **ioctl** method to access acceleration features. This library will handle functions which are commonly accelerated. The next suggest-string adds a few commands that are rather uncommon, but present in the ABC, which are accessed by the private area of the ioctls. The last library loaded accesses the ABC registers in an exported MMIO region. All the libraries are loaded in increasing order of precedence. The later ones override functions of earlier ones if they can do better. Please note this is not a static process—it can still change at runtime, if necessary.

Performance Considerations

When it comes to graphics performance, many people are afraid LibGGI will be slow because of the relatively high level of abstraction its extension libraries provide. Actually, this high level is necessary if we want to use all graphics cards at their maximum capability. Some high-end cards do have a truly high-level internal API. Having applications that use a low-level API would leave that part of the card unused.

On the other hand, in some cases it is difficult to decide which level of API to use. Consider a 3-D game. You can often do some clever optimizations based

on your knowledge of the scene. For a low-end graphics card, you might be able to calculate things faster yourself up to the rasterization level. With high-end cards you might be better off using OpenGL directly, because all calculations go to the card which does them faster than the host CPU.

This is a difficult problem, and actually the only good solution is to implement both and select one method at runtime.

Another ever-present problem is calling overhead. It is faster to use inline code than any kind of library. However, the biggest relative gains/losses are achieved with the very fast small operations such as **DrawPixel**. This is the primary reason we chose to implement **DirectBuffer** functionality. If the application knows the **DirectBuffer** format used by the graphics card, it can use its own inline code to bypass the calling overhead.

LibGGI should perform well over the whole range of possible applications and graphics cards, though specialized solutions might perform slightly better.

Available Applications

If you're considering using LibGGI either as a consumer or for programming your own applications, you might be interested in which programs are already available.

LibGGI has been designed for high speed graphics, so game designers are our primary customers. A lot of popular games have been ported to use LibGGI. **Descent** and **DOOM** are two of the more well-known ones. Using LibGGI, we managed to run Descent on a Linux-Alpha machine a few weeks after the source was released.

A common misconception about the GGI project is that we are trying to replace X. This is wrong. We are at a much lower layer than a windowing system, and have implemented some popular window systems on top of LibGGI. We have our own X server (**Xggi**), a viewer application for the VNC networked desktop, and the Berlin consortium is building its server on top of LibGGI. The existence of these servers together with the ability of LibGGI to display on them brings us to the next generation of interoperability.

Another broad group of applications deals with viewing files. LibGGI has the nice ability to view a JPG file on the console or in an X window, without the spawning application (such as **mc**) being aware that it is running on X or the console.

Most of the above-mentioned programs have been ported from other graphics APIs. All porters have told me that learning LibGGI was easy, and that after porting, the look of the program was improved.

If you are interested in LibGGI, you will want to know where to get it (see Resources). Our project home page provides many pointers and quite a few sources. LibGGI is available as releases from several major software archives like Sunsite and tsx, as daily CVS snapshots from our web page and its mirrors, as well as via CVS from our public CVS mirrors.

Several precompiled binaries are also available, which should be useful for the “pure user” who doesn't want to bother compiling LibGGI. Give LibGGI a try the next time you write a graphics application.

Andreas Beck studies physics in Düsseldorf. In his free time he enjoys adding new features to his favourite programs and operating systems. He can be reached at andreas.beck@ggi-project.org.

Archive Index Issue Table of Contents

Advanced search

Porting SGI Audio Applications to Linux

David Phillips

Richard Kent

Issue #53, September 1998

This article describes the process of porting a variety of audio applications from the SGI platform to the Linux operating system.

The process used to port SGI audio applications was reflective of Linux's own distributed developments—a truly international collaboration dependent on Internet communication. It is still a work in progress, with improvements and extensions to the software being created and contributed by programmers around the world.

Background

NoTAM is the Norwegian network for Technology, Acoustics and Music research located at the University of Oslo. Professor Oyvind Hammer of NoTAM wrote a series of applications designed to aid musicians and researchers in the analysis and representation of sound. Written originally for Silicon Graphics (SGI) computers, these applications utilize the graphics capabilities of the X Window System and make use of SGI's audio and sound file systems. Many applications offer basic and advanced editing features as well as sound file playback capability.

Preliminaries

Although not traditionally thought of as an audio platform, Linux already has several sound file editing and processing systems. Packages such as MiXViews, Snd, XWave and the CERES Soundstudio are available for audio recording, editing and playback. Many other packages can be found on the Internet. The Linux Soundapps page (see Resources), which I maintain, is a comprehensive and up-to-date list of Linux-based audio applications.

The editing capabilities of the NoTAM software are of a different nature: edits are performed on the graphic results of a Fast Fourier Transform (FFT) of a sound file. Explaining FFTs is beyond the scope of this article, but the results of a transform are usually depicted in a "spectral representation", i.e., a representation of frequency versus time. With the appropriate software, edits can then be made directly on the frequency content of a sound. Until recently, Linux had no such software, so porting the NoTAM applications to Linux was an attractive prospect.

Starting Out

When the NoTAM source code (made publicly available by Dr. Hammer and NoTAM) was examined, I noted that the graphics code consisted of plain X calls, and its sound support consisted of calls to the SGI-specific audio and audiofile libraries. Many of the applications were built with Motif. Since the necessary X libraries and LessTif (a freely available replacement for the Motif libraries) were already available, all that was needed in order to do the port were replacements for the audio and audiofile libraries. I contacted Dr. Hammer and asked him for permission to try porting the software to Linux, and with his gracious consent, the porting project began.

Scouting Around

Looking over an excellent web page dedicated to SGI audio applications (maintained by Doug Cook), I noticed a sound-file editor named **DAP** (Digital Audio Processor), written by Richard Kent. DAP uses the XForms libraries, so I inquired about the possibility of a Linux port. Richard informed me that he had already written such a port, and when I mentioned I wanted to port some other software written for the SGI to Linux, he graciously supplied the code used in DAP to replace the SGI audio and audiofile libraries and headers. The Linux versions of libaudio.a, libaudiofile.a and their associated header files are virtually direct "plug-in" replacements, meaning I could leave the NoTAM sources relatively intact.

Porting Begins

Armed with X11, Richard's replacement code and LessTif, I attempted my first port. I chose Dr. Hammer's **Sono**. This program analyzes a sound file and creates a PostScript sonograph of the spectral analysis. Since Sono does not display the graphics, instead relying on external display mechanisms, the port was fairly trivial, requiring only minor modifications.

With this first success, I moved on to another relatively straightforward program, **PTPS**, which creates a PostScript graph of a pitch-tracking analysis.

PTPS also compiled easily with only small changes, so I decided to attempt a more substantial port.

Ceres is an FFT-based program, but its design goes far beyond the simplicity of Sono and PTPS. Ceres renders the FFT results into a graphic display which can then be edited directly and saved as a new sound file. The challenge in porting Ceres was primarily in the X programming. Since no real-time aspects were involved, there were no problems with audio playback. There were, however, problems with the use of variable-length Xt argument lists which, in theory, must be terminated with a NULL entry. The SGI compiler and libraries did not appear to require this NULL; however, the Linux GCC compiler and libraries were stricter, and Ceres would fail with a segmentation fault upon opening if the NULL was missing. In addition, a problem with different maximum random numbers (**RAND_MAX**) between SGI and Linux caused a crash when using the Random Sieve transform. Once these two problems were solved, the porting of Ceres was complete.

I then decided to do an even more ambitious port, Dr. Hammer's **Mix** package. Mix is a 9-channel audiofile mixer with graphic waveform displays, graphic volume and panning curves, a scripting language for complex mixes, and real-time effects processing. (See Figure 1.) Obviously, audio playback capabilities are exploited to the full. I thought porting Mix would be by far the most difficult challenge, yet with Richard's replacement libraries (and some judicious code-cutting), I was quickly able to compile the Mix application, and Linux now has an excellent 9-channel, sound file mixer.

Figure 1. Mix Screenshot

Releasing the Packages

I placed the ported programs and their sources on the Music Technology Department's FTP server at Bowling Green State University in Ohio, I then notified Dr. Hammer of our successes (he was pleased we had achieved so much), and I also advised NoTAM that their software was now available for Linux. NoTAM obtained the packages and placed them on their server, making the applications easily available to everyone. I also sent notices to the Csound mailing list and comp.os.linux.announce to inform the Linux community of the availability of these packages.

Since the releases were made public, development has continued. Working from our source packages and Richard's library code, the Swedish composer Reine Jonsson has contributed a version of Mix which now handles the popular WAV format sound files (the original Mix, along with the other NoTAM packages, supports only AIFF format files), while reducing loading times and enhancing playback smoothness (a critical factor on my 486/120). A new

version of Ceres (see Figure 2) called Ceres2 is in development by Johnathan Lee and should be available in a Linux port by the time this article is published.

Figure 2. Ceres Screenshot

Further Development

Improvements can still be made: the applications ported so far are reasonably stable but will sometimes crash for no apparent reason. In some cases, not all of the original functionality is available, particularly if the package uses routines specific to the SGI's audio hardware capabilities. The Mix source code, for instance, includes calls to the SGI MIDI interface, but a replacement library for those calls has yet to be written. For now, I have had to disable the MIDI control code in the Mix sources. I have received a substantial amount of mail from users who have expressed interest in seeing more of this porting development, and my hopes are high that we will soon have a replacement for the SGI MIDI libraries to add to the audio and audiofile libraries already supported.

It must be mentioned that the NoTAM packages are not the only sources for high-quality UNIX audio-processing software available for possible porting. Guenter Geiger has successfully ported Paul Lansky's **RT**, another real-time, sound file mixer with excellent scripting capabilities. Work proceeds on ports of Paul Lansky's **Ein** (a DSP scratch pad), Mara Helmuth's **Patchmix** (a graphic patcher for the Cmix audio-processing language), and Russell Pinkston's **XPatchWork** (similar to Patchmix, but using the Csound language). Many other audio-related packages are available for Linux, and the interested reader should look at the Linux Soundapps web page for a continuously updated and comprehensive listing.

Final Thoughts

When I first used Linux I was thrilled by its possibilities, but dismayed by the lack of high-quality sound-processing software. Nevertheless, I was inspired by the availability of source code and the willingness of the Linux community to help develop audio applications. Since I am not a programmer, I relied on the skill, experience and advice of Linux users around the world. Like Linux itself, these projects were developed by a distributed collaborative effort, heavily dependent on the Internet for all communication, and built with freely available tools and libraries.

Thanks to Dr. Oyvind Hammer, Richard Kent, the LessTif developers and the XFree86 project, Linux audio software grows in quantity and quality almost daily. I encourage interested developers to contact me and let me know what they're working on or what they wish to work on. Many projects are waiting for

developers who would like to contribute their talent and interest to the rapidly growing Linux audio and music software base.

Technical Considerations

by Richard Kent

When I first heard of Dave's project to port audio applications from an SGI-based environment to Linux, I was very interested in becoming involved—particularly because the sheer dearth of audio applications for UNIX was the primary reason for programming the Digital Audio Processor. I initially implemented DAP for SGI-based systems, but shortly before Dave contacted me, I successfully ported DAP to run in a Linux-based environment. (See Figure 3.) This experience helped greatly when porting the NoTAM applications. This sidebar is intended to detail the three main technical considerations when attempting such ports.

Figure 3. Digital Audio Processor

Audio and Audiofile Libraries

Most, if not all, SGI audio applications make extensive use of the excellent audio and audiofile libraries supplied with IRIX. The audiofile library provides an API primarily designed to allow effortless loading and saving of AIFF audio files. The audio library is designed to allow straightforward audio input and output as well as control global audio settings. In order to make porting as painless as possible, replacement libraries had to be written for the Linux operating system.

The audiofile library was tackled first. Since this library simply has to perform file I/O based on the calls made, it was relatively straightforward to set up the necessary AIFF structures and to initialize, load and save these structures as necessary. However, because samples are read from and written to disk one sample frame at a time, this straightforward port of the audiofile library is relatively slow. In addition, only AIFF files are supported—compressed AIFF-C and WAV format files are not.

The audio library was a more demanding port, requiring extensive investigation into the facilities provided by the Open Sound System (OSS/Free) driver which is, by default, compiled into the Linux kernel. The basic process when using OSS involves opening /dev/dsp and either writing sample data directly to the device or reading from the device. In addition, opening /dev/mixer allows control over the global audio settings.

The Linux conversion basically sets up internal sample queues and provides facilities to transfer these sample queues to and from /dev/dsp. Most complications which arose were due to the many different sample formats (resolution and number of channels) supported by both the audio library and the OSS driver, thus requiring many different data conversions on input and output.

The resulting audio library on Linux is very much a “brute force” conversion and differs significantly from the SGI-based library, despite the almost identical API. The main difference is that the Linux audio library is not threaded whereas the SGI-based library constantly inputs and outputs sample frames from a cyclic queue in the background. As a result, the API user needs to be aware that samples must be constantly written to or read from the device to avoid audio glitches. Also, when finishing sample playback, blank samples must be written to the device to force the remaining sample queue to play before closing the device. The other main difference is that only one audio “port” may be open at any given time, due to the exclusive nature of opening /dev/dsp.

Compiler Differences

The default SGI compiler is quite different from **gcc**, which is the most commonly used compiler on Linux. More specifically, the SGI compiler is “relaxed” and not nearly as strict as gcc. This manifests itself in several ways, which must be considered when porting software from one platform to the other.

The most obvious difference is that explicit casting is often required on gcc to avoid warnings and errors which do not occur when using the SGI compiler. Two examples are shown here.

Default SGI compiler accepts:

```
int x = 3.2;
int *y = calloc (10,sizeof (int));
```

Linux gcc requires:

```
int x = (int) 3.2;
int *y = (int *) calloc (10,sizeof (int));
```

Correct link order is also more important when using the Linux gcc linker. The default SGI linker appears to place little importance on the order of link components (object files and libraries) when linking, as long as all “loose ends” are tied up by the end of the linking process. The Linux gcc linker, which I freely admit to not fully understanding, is much stricter and frequently requires

reordering of link components and sometimes even duplication of linked libraries.

Another major difference between the SGI default compiler and gcc arises when combining C and C++ files where the C files cannot, for syntactic reasons or otherwise, be passed through the C++ compiler. When using the default SGI compiler, the command for compiling both C and C++ files is CC, so there is no need to explicitly specify linkage specification using the **extern** C construct. When using the gcc development environment, the command to compile C files is gcc and the command to compile C++ files is g++; thus, the linkage specification must be specified when referring to C-based functions within C++ files, or else linking will fail due to C++ name mangling.

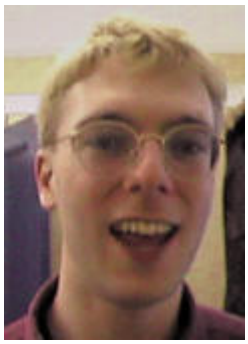
Variable Argument Lists

One final major difference between SGI and Linux development environments is that of variable argument lists for Xt and Motif function calls. The Xt toolkit provides the developer with basic GUI constructs which may be used directly to create a user interface. In addition, Motif and LessTif use the Xt toolkit to provide a higher-level API for constructing user interfaces.

Within these toolkits are functions which have a variable number of arguments, much like the standard **printf** system call. Unlike printf, these functions require a null entry to terminate the argument list. However, in the SGI development environment, these null entries are optional and SGI developers frequently forget to terminate the argument list with such an entry. This does not cause a problem on SGI-based systems, but if this same code is then compiled in a Linux environment, the resulting executable will almost certainly core dump upon execution. The fix is simply to add the missing null entries to the relevant calls.

Resources

David Phillips is a composer/performer living in Ohio. He has been involved with personal computers since 1985, when he first saw and heard a demonstration of MIDI music-making. Recent computer-music activities include an ambient composition for the artist Phil Sugden, lecturing on computer-music programming languages at Bowling Green State University, and maintaining the “official” version of Csound for Linux. Besides playing music and programming, Dave enjoys reading Latin poetry, practicing t'ai-chi-ch'uan, and any time spent with his lovely partner Ivy Maria. He can be reached via e-mail at dlphilp@bright.net.



Richard Kent is a professional C/C++/ActiveX developer currently working in Edinburgh, Scotland on traffic analysis and simulation software for both UNIX and NT. Richard has a very keen interest in the field of music technology and is the author of DAP (Digital Audio Processor)--a popular sample editor for Linux, SGI and Solaris operating systems. He can be reached via e-mail at rk@quadstone.com.

Archive Index Issue Table of Contents

Advanced search

Visualizing with VTK

James C. Moore

Issue #53, September 1998

A look at a new tool for visualizations of scientific data—VTK, an object-oriented visual toolkit.

Most scientists and engineers are adept at approaching and solving problems. If they use the scientific method, they may even get the right answer. However, analyzing results and measurements is often difficult because visualization tools are lacking. If your experience has been like mine, the tools for investigating data are either too specialized (**vis5d**), too weak (**plotmtv**) or too expensive (**AVS**). While good commercial packages exist such as Tecplot from Amtec Engineering, they often place restrictions (such as no remote displays in X) and constrictions on the user.

To solve this problem, three very intelligent men put their heads together (for nine months before coding began) and wrote The Visualization Toolkit (VTK). Will Schroeder, Bill Lorensen and Ken Martin have created one of the best systems available for performing scientific visualization. It is far and away the best value to be found today.

In this article, I will briefly describe what is required to obtain, compile and use VTK. The goal is to leave you with a sense of the scope of VTK and the level of commitment required to use it. You probably won't be able to immediately start creating visualization pipelines; however, you will have a good idea of the range of problems it is suited to solve and what will be required of you to solve them.

Overview

VTK is a collection of visualization “objects” which can be connected to make a visualization pipeline. VTK strictly follows the object-oriented model, whereby there is a hierarchy of objects, and any object which is a “sub-object” of another inherits the parent object's methods and properties. Objects are also broken

down into “classes” which represent the authors' best estimate of the most effective set of tools required to put together a visualization. The objects are broken down into 14 categories by function: Foundation, Cells, Datasets, Pipeline, Sources, Filters, Mappers, Reader/Writer/Importer/Exporter, Graphics, Volume Rendering, Imaging, OpenGL Renderer, Tcl/Tk and Window-System Specific. The user will most often be concerned with Dataset, Pipeline, Sources, Filters, Reader/Writers and Graphics and/or Imaging or Volume Rendering, though the other classes are implicitly used in most cases.

With these classes, we have the ability to construct a “pipeline” which reads or creates data, filters it as required, and finally renders it on the screen or exports the rendering to a file. While the classes follow the object model, the pipelines are procedural, which is most often needed when reducing data. The pipeline starts with a source (data), is operated on by any number of filters (even recursively) and is finally presented as output. The “data” source may be unstructured points, structured points, structured grid, unstructured grid, rectilinear grid or polygonal data. The class of data determines the types of filters which may be used to operate on the data, with the more structured data having the most filters available. For example, unstructured points may not be contoured, but they can be remapped to a structured points set, which can be contoured. Armed with these tools, all that is required to visualize almost any data is a sound approach to reducing it. With the ability to visualize data well in hand, the rate limiting step is now relegated to performance, and it can be a big issue. Datasets can easily get quite large or require a lot of computational effort to manipulate. VTK has tools to deal with these issues as well.

Using VTK

To start, I recommend the book *The Visualization Toolkit*, Second Edition, by Will Schroeder, Ken Martin and Bill Lorensen, published by Prentice Hall. It is an invaluable reference for understanding both the visualization process and VTK. After you've read the terse (yet complete) man pages, you'll understand why the book is needed.

All of the following examples were created using the Tcl/Tk bindings to VTK. These examples can also be created in C++, Python or Java; the latter two are relatively new to VTK, so your mileage may vary. Some examples were borrowed with permission from the VTK distribution, and all are biased toward reduction of computational data as opposed to imaging data or graphics applications.

Often, the first thing we ask to see when we have a large dataset is “Where is the data?” A simple enough request, but most tools will not easily give it to you.

Let's say we have smooth-particle hydrodynamics code which uses and generates unstructured points. For each point, we have the x, y and z coordinates as well as several scalar values (for now, tensor components will be considered scalars).

While not the most memory-efficient, one way to "see" the particles is to place glyphs at every particle position, scaled by the particle size. The visualization pipeline must take in the point data, create a glyph object and place one at each point location, scaled by the particle size. The set of glyphs must then be rendered on the screen. Listing 1 is the Tcl version of the code to do that, assuming you have read the point positions into the arrays **xpos**, **ypos** and **zpos** and the radius into **rad**.

Figure 1. Spherical Glyphs Scaled Relative to Particle Size

When this pipeline is run, a visualization window is opened on the desktop with a spherical glyph centered at the location of each point and a radius equal to the particle size (see Figure 1). Objects which are implicitly included in the scene but not listed above can be specified if required. These include lights, cameras and object properties. The implicitly defined objects are also accessible and controllable through their "parent" object; in this case, the renderer. In Tcl, a handy command is available from the VTK shell called **ListMethods** that informs you of all methods (and the number of arguments) available for any object. Adding the command **ren ListMethods** to Listing 1 would return the information that about 60 methods are available to you. After using this command on several objects, you will begin to see a structure to the methods and develop a sense of how the objects fit together.

With the addition of Tk entry boxes, all controllable attributes of all objects can be controlled interactively. However, changes to the pipeline will be seen only when the pipeline is re-executed with the new value and an **Update** is requested. This can be handled either by setting all attributes in a procedure called from the Tk interface or by attaching the method to the "command" argument of the widget that sets the value of the attribute. I recommend the former method.

The main access to the visible attributes of the scene is through the **Actor** objects and the **Mapper** objects. Attributes, such as visibility, color, opacity, specular reflectance, specular power and representation (wireframe, points, surfaces), are set with the **vtkProperty** object that is automatically created for the **vtkActor**, if one is not explicitly defined.

Now, let's say you want to evaluate a mesh created with an automatic mesh generator, and furthermore, you want to tag the cell with the smallest spacing.

Starting with the nodes and connectivity list, the “mesh” can be built by connecting the connected nodes with line segments and placing a geometric object at each of the nodes containing the shortest and longest connection. Listing 2 is a “quick and dirty” bit of code that took me about 15 minutes to write (well, maybe a little longer). It assumes the nodal positions are known, and their right, back and upper neighbors are known and stored in the arrays **i1tab**, **i3tab** and **i8tab**, respectively.

Figure 2. Mesh Nodes, Connectivity and Minimum Cell Dimension

The code in Listing 2 creates the visualization shown in Figure 2. This pipeline does not include the code to make the boundaries visible. We will cover that next. The key features of this pipeline are the multiple sources (mesh data, sphere) presented in one scene. The sphere is placed on the node with the nearest neighbor in one of the three coordinate directions mentioned above. The polygonal data represented by the **vtkPolyData** object called **mesh** consists of two-point polygons, i.e., lines. Polygonal data is often read in with a reader or created automatically by a source or filter such as the **vtkSphereSource** (Listing 1) or a **vtkContourFilter**. Notice that the mappers for the mesh and for the sphere are different. The mesh mapper takes the mesh directly as input, but the sphere mapper operates on the **vtkSphereSource**, which is not **vtkPolyData**. The reason for this is that the mapper *reads* **vtkPolyData** as input. The mesh *is* **vtkPolyData**. The sphere is a source which can send out **vtkPolyData**, when requested, as we do when employing the **GetOutput** method on the **vtkSphereSource**.

Satisfied with our mesh, let's look at some data. The pipeline excerpt in Listing 3 is based on the same mesh data as above, but includes methods to show the boundaries in the model and vector fields. [Complete listings for this article are available by anonymous download in the file ftp://ftp.linuxjournal.com/pub/lj/listings/issue53/3010.tgz.]

A lot is happening in the pipeline shown in Listing 3. First, the “mesh” polydata set gained two attributes: scalars and vectors (e.g., **SetScalars**, **SetVectors**). The vectors were created in a **vtkFloatVector** object. Their magnitudes were calculated and stored in a **vtkFloatScalar** field called **field**. The scalars are used by the mapper to color the vectors, and the vector data is used by the **vtkHedgeHog** (vector plotter) to create the oriented and sized vector glyphs. A separate pipeline is used to draw the surfaces of the object, and a 7-case switch is used to build the point connectivities for the surface panels per cell. It takes advantage of any connectivity which may exist on a given cell and builds a special type of polydata called “triangle strips”. Triangle strips allow *n* connected triangles to be created with *n + 2* points. The **vtkPolyData** must be told that the given cell array values are triangle strips in order to properly set

up the connectivity. This is accomplished with the **SetStrips** method, as compared to **SetLines** in the mesh example. The panels are made transparent by setting the opacity to .5, which allows the vectors to be seen. The color map for the vectors has been explicitly set to range between the minimum and maximum velocity magnitudes. By default, the mapping is red-to-blue from 0 to 1. The **SetScalarRange** method allows the range to be reset in the mapper. Notice the red vectors in the back left corner of Figure 3—an error is creeping in from the boundary and the location where it begins is very clear. Apart from verifying the correctness of the mesh, boundaries and boundary conditions, I can easily diagnose trouble spots in the calculation.

Figure 3. Vector plot of fluid velocities colored by magnitude and outer boundaries visualized with transparency.

Finally, these last two figures demonstrate some of the advanced features of VTK. Figure 4 is a BMRT-rendered (Blue Moon Rendering Tool) export from VTK. The complex shapes were built entirely from contoured implicit functions. Figure 5 is from the VTK examples directory and shows streamlines emanating from “seeds” that are located at a heater vent.

Figure 4. A BMRT-rendered view of a VTK scene. This visualization contains a complex combination of implicit functions, polydata and filters.

Figure 5. Plot from the VTK sample suite. This visualization includes streamlines flow colored by temperature.

While this treatment only scratches the surface of VTK's capabilities, you can begin to see the flexibility and power it affords the user. In addition to the features discussed in this article, VTK has objects for image analysis and manipulation, implicit functions, data transformation, data sampling, volume (solid object) rendering, memory management, texture mapping, data manipulation and exporting and more. Admittedly, the learning curve for becoming facile with VTK is somewhat steep, but it pays for itself many times over in saved time when doing complex analyses.

Obtaining VTK

The official source code release of VTK is available from the VTK home page at http://www.kitware.com/vtk.html. For the more daring, almost-daily beta releases are available from http://www.kitware.com/vtkData/Nightly.html. On the average Linux system, software required to compile and run VTK includes the following: C++, OpenGL (or Mesa), tkUnixPort.h (from the Tk source distribution), Tcl 7.4 or higher, Tk 4.0 or higher. If you plan to use the Python or Java bindings to VTK, you will need those packages as well.

Compiling VTK

The VTK source code is written entirely in C++, and as of version 2.0 with Linux 2.0.31 and either libc5 or libc6, it compiles successfully without error with Mesa 2.5 and Tcl/Tk 8.0. In the README file at the top of the distribution, the user will find all the instructions necessary to do the build. Here's what I did:

1. Obtained and compiled Mesa (easy).
2. Retrieved tkUnixPort.h from the Tk source distribution and placed it in the (vtk_top_dir)/unix/directory (I used Tcl/Tk 8.0).
3. Ran **./configure --with-mesa --with-tcl --with-shared --with-tkwidget --with-patented**
4. Edited user.make to find all the necessary support files.
5. Ran **make**.
6. Ran **make install** (optional; run only if you have the disk space).

Many more configuration options are available and can be seen by typing **./configure --help**. I had trouble with the Python and Java bindings. The build, as configured above, takes about an hour on a Pentium-Pro 200MHz machine.

Many examples are available to test the installation in the (vtk_top_dir)/[graphics|imaging|patented|contrib]/examples[Cxx|tcl|Python] directories. Most imaging examples require the vtkdata archive to also be located at the VTK home site. Graphics examples will, for the most part, run as is. For the C++ examples, compile with **make** and run. Tcl examples can be run by typing the following from the Tcl examples directory: **../../tcl/vtk *example_file*.tcl**, or, if vtk has been installed, **vtk *example_file*.tcl**. Examples employing the **TkRenderWidget** object cause a segmentation fault when using the XFree SVGA server, but work with the S3 server. (I haven't tested others.) Fortunately, **TkRenderWidget** is not required for any visualization pipeline; you can't embed the render window in a Tk window. However, this problem will likely be solved by the time you read this article.

James C. Moore is a Research Scientist for Applied Research Associates in Columbus, Ohio. His interests include numerical simulation of casting processes, gardening and losing money by fixing old Mercedes. He and Kim

(the real writer) have two daughters, Lorien and Kathryn. Jim doesn't do Windows. He can be reached via e-mail at jmoore@ara.com.

Archive Index Issue Table of Contents

Advanced search

Porting MS-DOS Graphics Applications

Jawed Karim

Issue #53, September 1998

Are you hesitant about porting your favorite VGA MS-DOS program to Linux? Using this tutorial and SVGALIB, porting will truly become a matter of minutes.

I first started VGA programming under MS-DOS, using the popular DJGPP C compiler. In recent years, this protected-mode 32-bit compiler, which is basically an MS-DOS port of **gcc**, has established itself as one of the preferred compilers in the MS-DOS game programmers' community. DJGPP was in fact the MS-DOS compiler of choice for idsoftware's game Quake. For the Linux console port of Quake, the Linux Super VGA library, SVGALIB was used.

When I first decided that I was going to port my own 3-D Model Viewer, **jaw3d**, from MS-DOS to Linux, it seemed logical to use the same approach. SVGALIB is very intuitive and allows me to easily maintain and further develop my 3-D Model Viewer for both platforms.

I found the easiest way to work with one set of source files for both platforms was to use preprocessor directives in places where different code was needed. Since I had already written and used DJGPP's low-level VGA and mouse instructions for the MS-DOS version, I simply added the equivalent SVGALIB Linux code in each instance, and separated the MS-DOS and Linux code using the preprocessor directive **#ifdef**. The following code snippet represents one of the many ways in which this can be accomplished:

```
#ifdef __DJGPP__
...
...
#endif
#ifndef __DJGPP__
...
...
#endif
```

__DJGPP__ is automatically defined by the DJGPP compiler, and is not defined by gcc under Linux.

An additional advantage of using SVGALIB under Linux is the fact that there is also a DJGPP version of SVGALIB. Let's try not to get confused at this point: SVGALIB is a graphics library that does some behind-the-scenes low-level VGA and mouse work for the user. Although SVGALIB was first developed for Linux, someone eventually released a version that worked with DJGPP under MS-DOS. Why not use SVGALIB for both MS-DOS and Linux? This would allow us to have 100% identical code for both platforms.

I don't recommend this approach, however, for two reasons. First, when I made speed comparisons of my 3-D engine between the two platforms, I noticed that when using SVGALIB with DJGPP under MS-DOS, graphics performance was sluggish in comparison with SVGALIB under Linux. Second, the MS-DOS executable was unnecessarily big because it had to be statically linked with the SVGALIB library. Using SVGALIB under Linux did not seem to present any problems. Due to the use of shared libraries under Linux, the executable remained tiny when dynamically linked, and graphics performance was actually slightly better under Linux than under MS-DOS. For the sake of performance and executable size, I found it best to use DJGPP's low-level instructions under MS-DOS and to use SVGALIB under Linux. This makes a difference, especially in a setting like 3-D engines, where every frame-per-second counts.

The advantage obtained from using the DJGPP port of SVGALIB is the fact that you can test your SVGALIB Linux code under MS-DOS, without having to reboot. Except for speed and executable size, both versions of SVGALIB behave identically.

Note that the DJGPP port of SVGALIB is still in beta, but I ran across only one minor problem and that was easily fixed. The file vgakeybo.h included with the DJGPP port of SVGALIB seemed to differ from the file vgakeyboard.h under Linux; therefore, making cross compilations was impossible in cases where keyboard code was used. The two files should be identical, of course, and the solution is to copy the Linux version of the include file over the DOS one.

The three compiler-specific code aspects are VGA, mouse input and keyboard input. If you have completed an MS-DOS graphics application, you may be using much of this code already and can quickly add on the SVGALIB equivalent code. On the other hand, if you do not have any previous graphics programming experience, you will find the code shown in Listings 1 through 4 to be very useful.

Listing 1. Include Files

Listing 2. Initializing the VGA Mode

Listing 3. Keyboard Code

Listing 4. Mouse Code

Copying a Buffer to Video

In the case of my 3-D Model Viewer, jaw3d, a complete frame is first rendered onto a buffer which has the same dimensions as the screen, and then copied to video memory all at once, allowing us to display frequently updated screens successively without any flickering. This is done as follows:

```
        memcpy (video_buff, image_buffer,
                DIM_X * DIM_Y);
        /* video_buff was initialized above */
        dosmemput (image_buffer, DIM_X * DIM_Y,
                0xA0000);
        /* 0xA0000 is the video memory in VGA mode
         * 13h */
```

Waiting for the VGA Retrace

By waiting for the VGA retrace, we are telling the program to wait until the monitor's electron beam reaches the bottom of the screen. Since there is a short pause before it jumps back to the top, it is a good moment to switch palettes without seeing “rainbow colors”. Thus, before switching palettes, we should wait for the VGA retrace as follows:

```
        while (!(inportb(0x3da) & 8));
        while ( (inportb(0x3da) & 8));
        vga_waitretrace();
```

Setting the VGA Palette

The following code assumes you have a character array of 768 values, representing the RGB values for 256 colors. For example:

```
char palette[768];
        where palette[0] = R value of color 0;
        where palette[1] = G value of color 0;
        where palette[2] = B value of color 0;
                ...
        for (i = 0; i < 256; i++)
        vga_setpalette(i, palette[i*3],
                palette[i*3+1], palette[i*3+2]);
        outportb(0x3C8,0);
        for (i = 0; i < 768; i ++)
                outportb(0x3C9, palette[i]);
```

Compiling

After adding SVGALIB code to the program, it's time to compile. Simply compile with the **-lvga** option to link the SVGALIB library. This library is preinstalled on most Linux systems; thus, if you experience problems linking it, you probably don't have it installed and should download it.

jaw3d was programmed by the author and is a Nullsoft Inc. product. Other cross-platform applications may be obtained at http://www.nullsoft.com/.

Resources

Jawed Karim is a freshman computer science student at the University of Illinois at Urbana-Champaign and works part-time at the National Center for Supercomputing Applications. His hobbies include programming and bicycle road racing. He can be reached at jkarim@students.uiuc.edu.

LINUX JOURNAL

A Tale of DXPC: Differential X Protocol Compression

Justin Gaither

Issue #53, September 1998

When you have a slow modem and want faster transfer rates, data compression with this program is the answer.

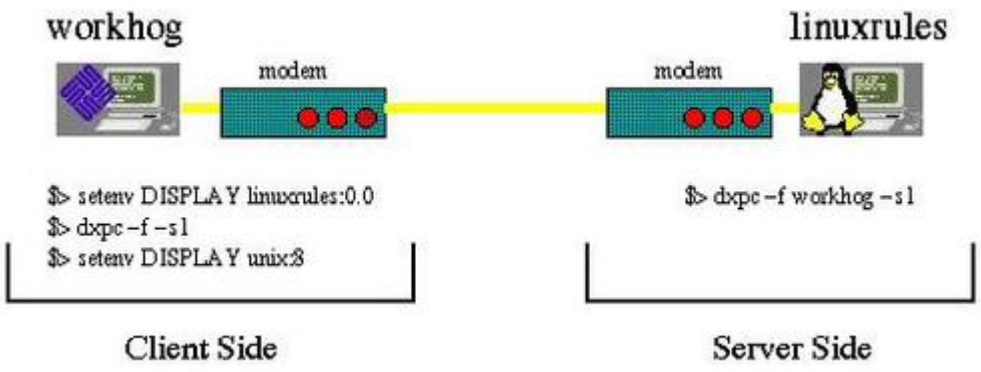
Once upon a time there was a frustrated engineer who needed a faster way to remotely display X clients on his home PC. He had a new daughter, and very much wished to spend time with her without driving back and forth to work. One day a fellow engineer told him about a fantastic little program called DXPC. Could it be true? Were all of his problems solved? No, but at least they were improved.

DXPC stands for Differential X Protocol Compression. It is a small client/server program that runs on both sides of a low bandwidth link (e.g., 28.8K modem PPP link), and is designed to exploit “features” in the X protocol to speed up the remote display of applications across the link. It is capable of compressing the X messages as much as 10:1. However, not all messages receive such great performance. Some messages are not compressed at all. On the average, you can expect as much as 4:1 compression. This sweet little jewel was originally written by Brian Pane and further developed by Zachary Vonler.

DXPC employs six different compression techniques. First, it strips unnecessary data fields. Next, it shrinks fields with a limited number of possibilities (i.e., Boolean). Using similar techniques, it shrinks fields that are typically small, while still handling the cases where they are large. The fourth method uses a cache on each side of the link to store different command types. With this cache, instead of transmitting full coordinates, it transmits a much smaller

differential value based on the previously sent command. Additionally, a cache of the last six deltas is stored so that it can do further encoding based on a pattern of deltas. Lastly, DXPC caches large blocks of data that are transmitted more than once (e.g., X resources). DXPC also employs a text compressor and an image compressor. For a more detailed description of these techniques, the authors have included a text file called DESIGN which describes how DXPC works.

DXPC is a client/server application, but the client and server are the same executable. The client side is the remote site or the site where the X applications are executed. The server side is the local site or the site where the X application is displayed. You must be able to compile DXPC and run it on the remote site and the local site. Fortunately, the authors have done most of the porting work for you. DXPC uses **autoconf** and will compile on most UNIX platforms right out of the "box". Additionally, there are ports to Win32 and OS/2. I compiled it for Red Hat Intel Linux (local) and Solaris 2.5.1 (remote) without a hitch. Well, one speed bump; I had to add the options **-lnsl -lsocket** to the LIBS line in the Makefile for Solaris. I have sent this little gnat to the authors, so hopefully it will be fixed in the 3.6 release.



Client/Server Configuration

Install

Perform the steps below on both your local station and on the remote site. Once you have completed this, you are ready to go.

```
tar xzvf dxpc-3.5.0.tar.gz
cd dxpc-3.5.0
./configure # optionally add --prefix=/home/bubba
make
make install
make install.man
```

Now the moment of triumph—let's look at an example. At work, you have a Sun box running Solaris. On this machine, named *workhog*, you normally run some expensive CAE (Common Applications Environment) tools that are not available for Linux. (If only these vendors realized the power of Linux.) At home, you have

a Linux box named *linuxrules* that you want to be able to use for more than Quake and Netscape. You have already compiled and installed DXPC on both machines, and are sitting at home wanting to do some work. You boot up *linuxrules*, and establish a PPP connection across that pathetic 28.8K modem to *workhog*. On the screen are two xterms, or rxvts. One xterm is attached to *linuxrules*, the other to *workhog*.

Inside the *workhog* xterm, type the following three simple commands:

```
setenv DISPLAY linuxrules:0.0
dxpc -f -s1
setenv DISPLAY unix:8
```

Then, in the *linuxrules* xterm, type this one simple command:

```
dxpc -f workhog -s1
```

Now you are ready to go. In the *workhog* xterm, start your very expensive CAE tool. Suddenly, it's as if your modem has turned into a T1 line. Well, not quite, but hopefully it is faster than before and fast enough to be useful. The **-s1** argument is optional and will report level 1 statistics on the compression ratio. There is also a level 2 statistics argument, **-s2**, which prints statistics on all message types sent.

Conclusion

The compression methods used by DXPC are compressions that cannot be done by hardware compression in the modem. In fact, I believe it complements other compression techniques to increase overall performance. The authors have done an excellent job of developing and maintaining a stable, easily compiled and easy to use program. I wish I had found it a year ago. Please note that if you use X authority with a .Xauthority file, some extra steps are necessary to use DXPC. These steps are covered in the README file distributed with the source.

Brian Pane informs me that they are preparing to release DXPC 3.6.0 soon. He has added compression for more X messages.

Resources

Justin Gaither is an ASIC designer for Alcatel Network Systems. He has been a Linux zealot for three years and hopes to enjoy his 15 minutes of fame. He can be reached at jgaither@aur.alcatel.com.

Archive Index Issue Table of Contents

Advanced search

Chess Software for Linux

Jason Kroll

Issue #53, September 1998

Once there was a time when chess software for the home was slow, weak and expensive. To find human opponents, you had to go to your local chess club. Today, the situations is different.

Linux offers a host of chess software that provides master strength computer opponents and analysis, and even an interface for playing against people all over the world via the Internet.

The strongest chess programs readily available for Linux (to my knowledge) are Crafty, Phalanx, and GNU Chess. Crafty is the strongest, though Phalanx and GNU Chess play at master strength (with fast hardware). Crafty is available via anonymous ftp from ftp.cis.uab.edu/pub/hyatt/ while Phalanx is available from the standard sunsite (sunsite.unc.edu/pub/Linux/games/strategy/) and GNU Chess can be downloaded from any GNU archive (prep.ai.mit.edu/pub/gnu/). The chess interface of choice (compatible with all three) is known as **xboard**, though a pretty 3D interface known as GLChess is available (the home page is http://nether.tky.hut.fi/glchess). The most recent version of xboard should also be available from any GNU archive, though an older release probably came with your Linux distribution along with GNU Chess.

Crafty

Crafty is the “long-time hobby” of Bob Hyatt, whose previous works include Blitz and Cray Blitz. Crafty is a very strong program and is constantly being enhanced. Though you could just download the Linux binary and use it as is, opening books and endgame databases are available from the ftp site, and add much to the playability and strength of the program. You have a choice between 1 MB (small), 30 MB (medium) and 74MB (large) opening books (or, if you like, the monstrous wall.gz which expands to 222 MB). For a basic installation, download the files crafty-14.13.linux, small.zip, start.zip, crafty.doc, crafty.faq and read.me, or the latest source if you want to compile it yourself.

Give Crafty its own directory (to store the book, game, position and log files), move the Crafty files there and unzip them. Execute Crafty (if you get a permission denied error, try using **chmod** on crafty-14.13.linux).

To create the opening book (book.bin) from the file small.txt to a depth of 60 ply (30 moves), type:

```
book create small.txt 60
```

The file books.bin should also be created in order to tell Crafty which openings it should (and shouldn't) play. The file start.pgn contains the necessary data. Just type:

```
books create start.pgn 60
```

After this brief “installation”, you should have a small, master strength chess program on your Linux box.

Performance can be maximized by allocating more memory to the hash and pawn hash tables. I have a 64 MB machine, so I set the options **hash=24M** and **hashp=10M**. Crafty configurations can be specified on startup; this means that you can include Crafty's startup commands in the resource file of your window manager for easier Crafty startup in X Windows.

Crafty can be run through xboard, with the Crafty-exclusive benefit of a splendid analysis mode (compatible with more recent versions of xboard) that allows you to move the pieces for both sides while Crafty rattles off analysis several moves deep (you can use analysis mode without xboard, but it's not as much fun). In order to start up Crafty through xboard, type:

```
xboard -fcp 'crafty xboard'
```

or you can specify more options such as:

```
xboard -fcp 'crafty xboard hash=12M hashp=5M'
```

Phalanx

Even though Phalanx, by Dusan Dobes, is the youngest of the three chess programs (it began in '97) it has managed to become quite imposing; in fact, it is not much weaker than some modern commercial chess software. Phalanx is also fun to play and good for variety, since Crafty, GNU Chess, and Phalanx all have different personalities. Phalanx should compile easily without any errors, and is then immediately functional. It has a small, default opening book, but creating your own from a PGN (pretty good notation) file is easy; instructions

are contained in the README file. Phalanx can be used through xboard by typing:

```
xboard -fcp phalanx
```

GNU Chess

GNU Chess is *the* classic chess program which has been around for ages on a number of platforms. It is also rather strong and quite fun to play. GNU Chess came with my Slackware and Red Hat installations, and I imagine it comes on other Linux distributions as well, so you may already have it on your machine. It is very fun for blitz chess, especially since with its default opening book it is prone to making original moves which may or may not be very good (this doesn't matter since it wins anyway). GNU Chess can be played in a console or under the X Window System through an interface. When xboard is executed, by default it loads up GNU Chess and prepares for blitz games of 5 minutes per side. You can, as usual, specify startup options; the xboard man page contains the details.

Chess on the Internet

xboard can also be used to play chess via the Internet, which is an excellent way to find fun opponents of all skill levels.

In order to use xboard to play on an Internet chess server, first set up a SLIP/PPP connection, and then to get to the main free U.S. server, type:

```
xboard -ics -icshost freechess.org
```

This will connect you to the Free Internet Chess Server (FICS) through port 5000 and open an xboard display so that you are ready to both observe and play games. You will first need to log in, of course, and for this you should choose a handle to use as a guest until you think of one especially clever. To get a list of the games in progress, type:

```
games
```

Then to test the interface, try typing:

```
obs 6
```

to observe game 6 (assuming it exists). Your interface should work and you should see some pieces moving hither and thither; type “unobserve” when you've seen enough. The commands are quite simple, and the online documentation is thorough, but if you need help getting started you can ask other players in channel 1. A great many chess servers exist throughout the world, including Grandmaster Dzindzichashvili's chess.net, the (commercial)

Internet Chess Club chessclub.com, and the main European free server eics.daimi.aau.dk; I imagine any of these would be happy to have new players.

Conclusion

Those of us who were made to suffer at the hands of Sargon in the days of the Commodore 64, tormented by the malicious characters of BattleChess, or politely dismantled by Chessmaster, now have the pleasure of choosing a variety of master strength programs to defeat us on our modern machines (the struggle to prove a consistent match to the ever-improving Crafty is being made into a movie, apparently to be called *Searching for Holes in Crafty's Opening Book*). However, today's computers also provide the frustrated chess player with relief that computers did not offer when the Commodore 64 was king: Internet chess, featuring human players, complete with mistakes and oversights that computers don't make anymore.

Jason Kroll is a student of economics at the University of Washington. He likes music, computers, and chess, and thinks that Linux is the best thing to happen to computers since monitors (or at least since the Amiga). He can be reached via e-mail at hyena@u.washington.edu.

Archive Index Issue Table of Contents

Advanced search

LJ Interviews LDP's Greg Hankins

Marjorie Richardson

Issue #53, September 1998

With the next Atlanta Linux Showcase (October 23-24) looming on the horizon, I decided it was time to get in touch with Greg Hankins, coordinator of the show and maintainer of the Linux Documentation Project.

With the next Atlanta Linux Showcase (October 23-24) looming on the horizon, I decided it was time to get in touch with Greg Hankins, coordinator of the show and maintainer of the Linux Documentation Project. We “talked” by e-mail about him and the show on June 1.

Margie: Let's start off with some personal information. Tell us about where you live, go to school, etc.

Greg: I live in Atlanta, GA, where I have been since 1988. Before that, I spent most of my life in Germany where I went to a German school through the 10th grade. When we returned to the U.S., I began the 11th grade in high school and decided that the only thing I wanted to do was to go to Georgia Tech (a technical university located in Atlanta) to study computer science. Ten years later, I'm still at Georgia Tech—I received my bachelor's degree in 1996 and I'm now half-way through the master's program.

I've been into computers since I was about 12 years old, when I started playing on our Apple][+. I got started with UNIX around 1990, when I instantly recognized it as the Right Thing. In fact, I even bought an old AT&T UNIX PC so I could have a home UNIX box. I started using Linux in the spring of 1993, and I still have the set of floppy disks I used for installation (SLS 1.01 distribution, I think).

In real life, I'm a Network Engineer at MindSpring Enterprises, an ISP with nation-wide coverage. My group is responsible for the daily care and feeding of our WANs and LANs, upgrading and expanding our network and evaluating new networking technology. I am fortunate to also have a wonderful girlfriend,

who somehow manages to keep me from spending all my time in front of a keyboard.

Margie: What do you for fun—do you have time for fun?

Greg: With school, work, Linux projects and trying to have a life, I do keep quite busy. Linux is supposed to be fun, so I guess all my Linux projects count as fun.

One of my hobbies is enjoying beer. I am an avid beer enthusiast, and I maintain a constantly growing collection of self-quaffed beer bottles and paraphernalia—at present, over 450 unique bottles. I attempted home brewing a few times and have even tossed around the idea of becoming a certified beer judge.

I also enjoy British humor such as Monty Python, Young Ones and the like, science fiction such as Dr. Who and Star Trek, computer history and old computers (that would explain the PDP-11 in my bedroom) and bad puns. I'm also fascinated by anything related to computers and high-tech toys—in that aspect I'm pretty much a standard geek.

Margie: The Atlanta Linux Enthusiasts seems to be a very successful user group. Tell us a bit about how it got started.

Greg: Well, it all started in 1994 with the “gtlinux” mailing list, a list we set up for students at Georgia Tech to discuss Linux. Some of the people from the list decided to form an Atlanta Linux users group to include students, professionals and enthusiasts in the metro Atlanta area. A posting to comp.os.linux.announce attracted ten people who wanted to help, and on December 15, 1994 the group was founded. We named it the Atlanta Linux Enthusiasts (ALE). Over 100 people showed up at our first meeting in January, 1995.

We intentionally have no formal charter, membership or organization. This keeps things simple and seems to work well. All people have to do is show up at the meetings—that's it. We decided on a meeting structure similar to the Atlanta UNIX Users Group (AUUG) meetings, which have been going strong since mid-1980.

Margie: What sort of programs does the group put on to keep people interested and coming to meetings?

Greg: We simply put on exhilarating monthly meetings and Linux conferences! Each month we have a speaker for our meeting. We try to vary our topics somewhat, in order to appeal to all levels of users. If we can, we alternate monthly between “new user” and “advanced user” topics. There is also a free-

form session at the end of the meeting where anyone can ask a question of the group. This gives people a place to go for help, as well as information. Most of our talks are given by ALE members; we have very few talks about commercial products given by vendors. We typically have around 50 people at our meetings —many people are regulars, but I do see new faces each time.

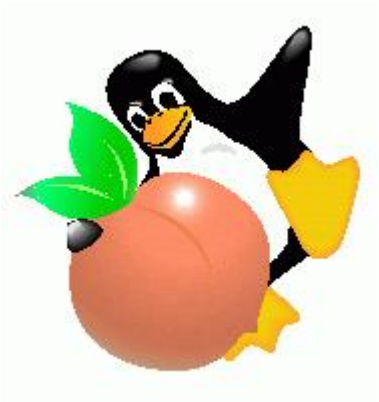
From time to time we've also had free pizza and Coke at a meeting or put on fundraisers (for example, I sold Dr. Linux and Red Hat CDs a few times). Door prize drawings are frequent, as many vendors send us CDs and books to give away. We will probably be doing an install fest in the next few months.

We also maintain a mailing list for our group. People use the list to ask questions about their Linux problems and to discuss Linux and related topics. There are over 200 people on the list, many from outside Atlanta and even some from outside the U.S.

Margie: Does ALE appeal to one group of people, such as students, more than another?

Greg: I'd say that we have a very good mix of professionals and enthusiasts in the group as well as quite a few students. We happen to meet on the Georgia Tech campus, but that hasn't affected the attendance mix in a big way. We certainly encourage everyone to attend and don't discriminate in any way.

There is a local Georgia Tech group, but they focus more on specific problems the students might have, such as using the dorm networks. A group at Emory University is forming for the same purpose.



Margie: Last year ALE put on a very successful Linux exposition, the Atlanta Linux Showcase, and is planning to do it again this year. Can you tell us some of the reasons last year's show was such a success?

Greg: Yes, we did put on a very successful Linux show last year. It was an all-volunteer-run conference and trade show organized entirely by our users

group. The thing that made it possible was an extremely dedicated core group of people who gave hundreds of hours of their time to support something that we passionately believe in—Linux. It was also to our advantage that we had an extremely diverse group of people who volunteered. For example, different people had prior experience with trade shows, accounting, printing, graphic design and other useful things.

We gave everything from time to loans from personal funds in order to make this event work. Linux International also helped us; in fact, it was Jon “maddog” Hall who first approached us with the idea of putting on a “small” show—little did we know what it would grow into. We basically had no capital and no legal organization that could sign contracts. Linux International provided critical support in those areas—we provided the hard work.

I think we had a combination of good conference programs and vendor exhibits, as well as a great location for the show. Atlanta is a great city for conventions. We have the facilities, a big airport, good transportation and many things to do and sights to see, which provided people with entertainment in the evenings. I think people had fun. It was great to see so many people in one place all talking about Linux.

Since then, we have taken the profits of last year's show and invested them in the next show. We formed a small corporation to give us legal standing and protection and have also been dealing with taxes and getting approval as a credit card merchant. Many rules and regulations exist which complicate things, and an enormous amount of determination and effort is required to make things fall into place. We're learning how to run a not-for-profit business, all in our spare time!

Margie: Any disasters to report?

Greg: I'm pleased to report we really didn't have any disasters, other than a noticeable lack of sleep on our part. The show was organized in about four months, which is a very short amount of time to organize any size conference or trade show. Many large conferences are planned years in advance, and the time frame in which we organized the show presented a few problems of its own. For example, booking exhibit space is almost impossible with only a few months' notice. We completely missed advertising deadlines and were too late in many cases to even get on upcoming event calendars. We've been planning our next upcoming show for over a year.

Another problem was dealing with money. Since we were an unofficial organization, we could not accept credit cards and thus could only accept cash or checks from conference attendees. We also had to put our checking account

in the name of one of our organizers, instead of the name of the show. This year, we have fixed both of these problems.

Margie: What are the plans for this year's show? Speakers? Highlights? How many people do you expect to come this year?

Greg: Our plans for this year include two days of technical and business talks about Linux, free and Open Source software, and vendor exhibits. Dr. Michael Cowpland, President and CEO of Corel, will be our keynote speaker. Jon “maddog” Hall, Bruce Perens and Eric S. Raymond are among our initial list of speakers. We're also planning BOFs (Birds of a Feather), a fundraiser dinner and a terminal room, which will be a great place to meet other people in the Linux community. The exhibits and activities will all be free, and the conference sessions will be reasonably priced with pre-registration and student discounts. By the time this interview is printed, we will have our on-line registration system running.

One of the highlights is our “Linux in Action” booth. This booth is staffed by ALE members, and it's a great place for people to use Linux hands-on. We demo a wide variety of hardware and software running Linux with theme areas such as “servers”, “windowing systems” and “productivity tools”, showing people what Linux can do. Attendees are free to use the machines and software packages and ask questions of ALE staff. Last year we had Linux running on over 15 Intel, Alpha, SPARC and PPC boxes, using five Linux distributions and loads of software. It was quite a popular attraction.

This year our goal is to double the show size. We are running ads in *Linux Journal*, *Boardwatch* and *Sys Admin* as well as launching an electronic ad campaign. We're planning for 1000 to 2000 attendees. We had 500 people attend last year with no national advertising and only a few months of electronic advertising, such as postings to c.o.l.a and contacting users' groups. Our show also follows NetWorld+Interop, one of the largest networking trade shows in existence. We hope to draw some of the attendees to our show in the same way that we followed COMDEX last year.

As for exhibitors, our goal is to fill 40 booths with vendors showing off Linux and related software, hardware, CDs, books, shirts and whatever else they wish to bring. Last year we had 25 vendors on the show floor, a record number at that time for a Linux show.

Margie: Anything in particular that you plan to do differently?

Greg: Yes, a few things. We're excited about the amount of time we have to plan this show, since we basically started planning it right after the 1997 show ended. With over a year to prepare, we're going to be able to do a lot more.

One of the biggest things we learned was that Friday/Saturday shows always do better than Saturday/Sunday shows. We had great attendance on Saturday last year, but Sunday was noticeably slower. This year our show starts on Friday and actually overlaps with NetWorld+Interop that day, which will be a mere three blocks away.

We also learned many small lessons. We met with each vendor that exhibited last year to find out if they had any comments or advice about the show. We learned things through experience, what worked well for us and what we could have done differently. There truly is no substitute for experience.

Margie: ALS sounds like it will be a fun and worthwhile conference for those who attend. Let's move on—tell us a bit about the Linux Documentation Project.

Greg: The Linux Documentation Project (LDP) was started in order to write documentation for the Linux operating system. According to the collective memory, it was started sometime in 1992 by Michael K. Johnson, Matt Welsh and Lars Wirzenius.

The overall goal of the LDP is to write documents that cover installing, configuring and using Linux. For information about the LDP, visit the LDP home page at http://sunsite.unc.edu/LDP/. You can find all the documentation in the LDP collection, as well as many useful links and information. The LDP home pages were some of the first Linux-related pages on the Net when Matt started writing them in 1994, and they have accumulated a lot of information since that time.

We have four basic types of documentation: guides, HOWTOs, man pages and FAQs. Guides are entire books on complex topics; for example, *Linux Installation and Getting Started* and the *Linux System Administrators' Guide*. HOWTOs are detailed “how to” documents on specific subjects, such as networking, SCSI or hardware compatibility. The man pages as well as many FAQs, including the Linux FAQ, are also produced. We have a few special documents that you can find on-line on the home page, such as *Linux Gazette, the* Linux Kernel Hackers' Guide and certain HOWTOs.

Many translation projects and non-English LDPs have been formed and more are starting. Links to these projects can be found on the LDP web pages. We also support them by getting some of their work archived on sunsite. This way

the documentation is easily accessible, and also gets distributed on CD archives.

Margie: What is your part in the LDP?

Greg: I joined the LDP in the fall of 1993 when I started the Serial FAQ after many frustrating hours of trying to get **getty** running on my box. This FAQ later became the Serial HOWTO, which I maintained until a few months ago. I had to give this up recently due to a lack of time.

I took over the HOWTO coordination from Matt in 1995, and managed the HOWTOs until April of this year, when I found a new victim/volunteer to take over for me. I decided to give this part of the LDP up too, again due to lack of time and a lack of enthusiasm.

Now I'm pretty much responsible for the LDP web pages and the overall LDP coordination as well as acting as a point of contact. I also maintain the /docs directory on sunsite and whatever else happens to fall into the documentation area.

Margie: Are there particular guidelines for submitting documentation to the project?

Greg: The most important thing to remember is to contact us *first*, and to get approval if you are interested in contributing to the LDP. In order to coordinate the documentation effort, we need to be aware of all the work different people are doing. This way, efforts are not duplicated and wasted. I have had to turn away several submissions because they were duplicated or, in some cases, not appropriate for the LDP.

We have set a standard of using LaTeX for the LDP Guides (large book-like references), and using SGML for the HOWTOs (short, specific “how to” documents). Currently, a package called “SGML Tools” is used to take the SGML source and produce PostScript, DVI, HTML and plain text output. All submissions must follow these standards so that we can provide a common look for the formatted documents and effectively manage the sources.

Margie: What other Linux projects are you involved in? Do you do any development?

Greg: No, I don't do any development. For some reason, I dislike programming so I help out in other ways. The LDP and ALE/ALS are my primary Linux projects, but I have also been known to help maintain sunsite's archive, and I have reviewed a few Linux books here and there.

Margie: What do you think is the most exciting project happening with Linux today?

Greg: I think the most exciting thing is that Linux is finally being recognized as a viable alternative and contending OS. (Of course, we knew it all along.) It is essential to have an alternative to Microsoft—I'm not interested in using the square wheel they re-invent every few years. 1998 has been a great year for Linux; just look at all the attention it's been getting recently. The trade magazines (both on-line and printed) have been full of Linux reviews and stories. For example, InfoWorld awarded the “1997 Best Technical Support Award” to the Linux community. It's good to see all the hard work by the developers and commercial companies paying off.

I'm also excited to see a lot of projects that have the goal of making Linux easier, more productive and more fun to use. Projects such as GNOME, GIMP and Linuxconf, to name a few, are providing Linux with some killer applications and tools. The Linux Standard Base (LSB) being set up by Bruce Perens should also be an interesting project.

Margie: Any parting words?

Greg: Just a couple of e-mail addresses and a URL. I can be reached via e-mail at gregh@sunsite.unc.edu. I'm always interested in comments about the LDP web pages or about the LDP in general. The HOWTO coordinator can be reached via e-mail at linux-howto@sunsite.unc.edu.

If you are interested in the 1998 Atlanta Linux Showcase or the Atlanta Linux Enthusiasts, visit our web site at http://www.ale.org/.

Margie: Thanks, I'll take a look today.

Archive Index Issue Table of Contents

Advanced search

Migrating to Linux, Part 2

Norman M. Jacobowitz

Jim Hebert

Issue #53, September 1998

We continue with our look at converting an office from a commercial operating system to Linux.

Welcome to the second article of our three-part series on migrating to Linux from a commercial operating system. Our first installment (in August) discussed many reasons why a non-technical, small office or home office (SOHO) end user might abandon their commercial OS and adopt Linux. This month, we'll explore several pointers SOHO Linux users may find helpful in making their migration safe, comfortable and productive. We'll also investigate some of the software tools, both commercial and Open Source, that SOHO users may find useful. Finally, we'll discuss interfacing and sharing files with our friends, clients and colleagues who have yet to see the light and join the Linux camp.

We assume the reader has one of the many fine Linux distributions installed and working. If you have not yet taken the plunge and installed a Linux system, please read on anyway. Hopefully, every current or potential SOHO Linux user will gain a more complete understanding of what it takes to use Linux in a SOHO environment.

Do You Have to Become an Expert?

One obstacle preventing SOHO users from considering Linux is the “hacker mystique” surrounding the OS. To many new users, Linux has the image of an expert's paradise, a playground for gurus only. We did much to explode that myth in our first article. Still, for many potential Linux users, the unanswered question remains: *how hard and time-consuming is it to maintain a Linux system in a SOHO environment?*

Once your system is installed and configured, you may discover that maintaining it is not that hard. You will not have to become an expert in system administration, nor will you necessarily have to spend much time logged in as root performing administrative tasks. In fact, you may end up spending a bigger percentage of your time actually working rather than dealing with the system. Is maintaining a SOHO Linux system any easier than other operating systems? Not necessarily, but many find that once they have developed a core of basic competencies, running their SOHO Linux systems is no harder than maintaining any other OS.

The Tasks at Hand

In order to gain confidence in ourselves and our Linux systems, SOHO Linux users need to be proficient in several key facets of system administration. The minutiae of all the various commands and tasks are beyond the scope of this article. Therefore, we will focus on those we feel are most important to SOHO users: maintaining software, performing regular backups and, in case of emergencies, boot/root disk use.

What is System Administration?

For the SOHO Linux user, basic system administration means all of the work done to keep the machine up and running smoothly, such as installing/ upgrading software and removing old files—for more accomplished users, it may mean compiling a custom kernel. Notice that our needs and responsibilities are considerably less than those of an administrator looking after a network of servers and workstations. Here are several key pieces of advice which you may find helpful:

- Be self-sufficient—invest in at least one decent Linux manual (see Resources). Locate and bookmark some of the many Linux documentation sites on the Internet. Read all available documentation before posting questions on the newsgroups. These suggestions may seem obvious and elementary, yet anyone who reads the newsgroups knows how many well-documented, easily-solved problems are repeated over and over again, wasting a lot of valuable time and energy for both the posters and those trying to help them. On the other hand, don't try to learn everything—there is just too much to know. Keep your Linux information resources handy and use them as a reference library. Whenever you are stumped or run into a problem, have your resources available and know how to use them. Especially at first, it is better to concentrate on “how do I look it up quickly” rather than trying to memorize individual commands.
- Keep an eye on the updates and recent developments going on in the Linux software community, in particular as they pertain to the distribution

you are using (see Resources). You will occasionally need to install or upgrade software as bugs and security problems are detected. You don't need to spend hours every day reading about Linux. One reasonable schedule is to spend at least an hour every 10 business days cruising the relevant web sites for recommended software updates and new tools that may help you get your work done. The kind and patient souls who read the newsgroups will thank you for reading about these updates before posting problems.

- Once you have your system installed and set up to your liking, do not log in as root unless absolutely necessary. Again, this is an elementary rule that Linux gurus usually follow, but SOHO users need to be wary and avoid this pitfall. It's truly hard to cause a full system crash in Linux—unless you are foolishly mucking about while logged in as root.
- Take the time to learn a little about the **bash** shell. Most Linux manuals have at least some introduction to bash and various shell commands, but a more comprehensive look may be in order. Knowing bash well can save you time. When tinkering with bash, remember point number one above: don't bother memorizing every command, but do keep your manual handy.
- After you have learned a bit about the system, made reliable backups and are feeling comfortable with Linux, you may want to try compiling a custom kernel. This exercise will teach you a lot about the way Linux works and may make your machine run a little faster than the generic kernels shipped with most distributions. The guide published in *Linux Journal*'s November 1997 "Kernel Korner" is one of the best concise guides to kernel compilation available. Read it, and if it seems to make sense, go for it. It's a fun and enlightening way to learn more about your system.
- Trust Linux and the people behind it. It's a powerful, reliable tool for the small or home office. The thousands of developers worldwide who work on it have seen to that. After all, that's why we are migrating, right?

Why Bother with Backups?

Here's a personal story from one of the authors that illustrates exactly why we all *must* make regular backups. I'll never forget it. It was late at night, and I was on a tight deadline, cranking out a major project for a client. Then, I heard it. A rattling noise, like a backgammon player shaking a cup of dice. Yes, my hard disk had just croaked—I was hearing the heads skipping across the platters in my hard drive. The machine would not reboot, nor could I access any data on the drive. After a few seconds of panic and terror, I tore open my desk drawer and cradled what at that moment was my most prized possession: a recent tape backup of that entire hard drive. The next morning was a flurry of intense activity. I notified clients of a brief delay. I contacted my computer manufacturer (the disk was under warranty) and demanded a new hard disk—

of course, it would take 10-14 days. I ended up running out and buying a new disk, installing it, reinstalling all my software and restoring most of my precious work from that tape. Eventually, the manufacturer replaced the dead disk.

The moral of the story? Disasters can and do strike ordinary people doing ordinary things on a computer. Your responsibility is to be prepared. In my case, the hard drive crash was softened from a disaster to an inconvenience, all because I had that backup tape. Fortunately, Linux comes complete with a set of utilities that make backups safe, easy and reliable. Plus, several commercial and free software options exist to automate and simplify the backup process. Ideally, backups should be performed daily. In reality, doing a full backup about twice a week is a reasonable schedule for most SOHO users. Plus, if possible, store every other backup off-site. Even if your spouse or parent takes one tape or disk to work and brings back the other one, you are reducing the risk of losing your backup to fire or theft. Never leave your backup media sitting in the backup device.

If you don't have access to a backup program, Linux has several options to tide you over until you get one. Probably the most used is the **tar** command, which you should have at least an understanding of, even if you end up using a different backup option. Spend some time looking at the tar man page. Check your Linux manual; most give a fairly thorough analysis of this key backup option. At the very least, until you have a more reliable backup routine, copy key files to a floppy using the **cp** command.

Boot/Root Disks

Assume it's 9 AM on a weekday. Your client expects delivery of a key project by noon. You finished it last night at 11:30 PM, after hours of revision and several pots of coffee. All you have to do now is boot up your machine, print out a final copy and fax it off. Except for one little problem: you machine refuses to boot. Only three hours until your deadline—what now?

Linux will rarely choke, provided you are not tweaking with the system or mucking around while logged in as root. However, some hardware failures can sneak up on you. In any case, you must be prepared to act when your machine doesn't want to boot. That means having and knowing how to use a good set of Boot/Root disks, also known as rescue disks (see Resources).

The Boot/Root disk may one day turn out to be your best friend. While a comprehensive set of instructions are beyond the scope of this article, you *need* to understand how to use them. Many distributions come with a set of boot disks that double as emergency disks, or you can download a pre-built set. In any case, get them and know how to use them before a disaster strikes.

Getting Stuff Done

Now that we have covered guidelines for basic system administration, it's time to think about doing some work. Fortunately, the last several months have seen an explosion in the amount of productivity software—both free and commercial—available for Linux users.

As for the Open Source versus commercial software debate, the choice boils down to the preferences and budget of the user. Most users will end up with a hodgepodge of both free and commercial software on their systems. SOHO users, additionally, have their limited time and accountability to clients or colleagues to consider when making their choice. Our advice is to thoroughly check the capabilities of any software you plan to use.

Let's look at an example. Say it costs you $100 to purchase a piece of commercial software and you figure it will take four hours to install, configure and gain familiarity with the product. Now compare that to an Open Source product. While you pay nothing for the software, it may take you ten hours to install and become comfortable using it. Which product do you choose? It's up to you and your priorities. Just remember: Open Source does not always mean free of expenses, while commercial software can never be automatically assumed to be easier to use or of higher quality. Judge each package by its individual merits and make the right choice for yourself.

Regardless of your choice, we believe it is up to us as SOHO Linux users to provide positive feedback to both commercial developers and the kind souls who develop Open Source packages. Linux is the best SOHO/workstation OS on the market right now, but most commercial developers have yet to embrace it with the support it deserves. On the other hand, we can't expect free software developers to fill all the voids left in the Linux software library. By rewarding those commercial developers who do port their software to Linux, we can encourage others to do so; and by using Open Source software, we encourage the type of cooperation that has made Linux the great OS it is today.

Does this mean any one development model is the best option for SOHO users? No, we are merely suggesting that if you do choose to use commercial software for Linux, please be sure to obey the developers' licensing restrictions and give positive feedback or bug reports on the product, if applicable. If you are using Open Source packages, please consider contributing to such development efforts, by donating your own time or perhaps even financially, if you can.

Applications . . . Applications . . . Applications . . .

Too many useful Linux applications exist to list them here. For the SOHO Linux user, a few key applications are available which do bear mentioning in this context. As for the other possible software tools, we should remember number two on the list of recommendations made earlier: know where to find good catalogs of Linux software (see Resources).

One of the biggest news items to hit the Linux community recently is the announcement from the Corel Corporation. Corel has committed to porting its productivity applications to Linux. For those of us who use our Linux boxes for earning a living, this is indeed welcome news. Corel's announcement, along with the continuing evolution of other Linux software, means that most Linux users may no longer have any reason to boot another OS.

Most SOHO Linux users will do the vast majority of their real work while in the X graphical environment. While there are at least a dozen useful window managers (see Resources) for Linux, the two user environments making the biggest headlines right now are the K Desktop Environment (KDE) and the GNU Network Object Model Environment (GNOME). A commercial option is the Common Desktop Environment (CDE). While many developer types and other gurus have different reasons to prefer one over the other, we SOHO users will most likely end up basing our choice once again on personal preference.

When you mention productivity, especially for SOHO users, the first thing that comes to mind is a good, full-function office suite. Right now, the two “biggies” in the Linux community are Applixware and StarOffice. Corel's port of its office suite will add a third option. As for e-mail and web surfing, Netscape's decision to open the source code for its Communicator 5 is perhaps the best news for the SOHO user. Until that product reaches maturity, Netscape's Communicator 4.0x will meet most users' needs.

Soon, we can look forward to at least one full-featured financial management program—a la Intuit's Quicken line of software—known as GNU Cash. As for graphics and graphic manipulation, we have the GNU Image Manipulation Program (GIMP) which rivals Adobe's PhotoShop. Corel has also ported their CorelDraw program; as they port the rest of their commercial applications to Linux, we hope they will take the time to update the package.

Sharing Data

While trying to make up your mind about which software you'd like to use, pay particular attention to whether or not you will need to share files with your non-Linux friends or colleagues. This is especially important for users of the various office suite tools. For example, will you be able to share and save your

files in a de facto standard format such as Microsoft Word (.doc)? Or will you be forced to save and share your files in another format like Rich Text Format (.rtf)? Either way, you need to be sure your clients and colleagues will be able to use the files you produce for them.

Another key skill for integrating with non-Linux users is the ability to mount, read from and write to MS-DOS, VFAT and HFS floppies and other removable media. This will enable you to share floppies with Microsoft and Macintosh users. Check under the **mount** command in your Linux manual and read the man page for mount and also the MTOOLS and HTOOLS utilities.

Until Next Time

We've talked a little about basic system administration, software for the SOHO user and sharing files with our colleagues. Let's sum up a few of the key points:

- Make yourself as self-sufficient as possible by owning at least one comprehensive Linux manual, bookmarking and regularly visiting major Linux software web sites, judicious use of relevant IRC channels, and refraining from posting to newsgroups until you have exhausted other avenues of help.
- Make reliable, frequent and regular backups, using any supported removable media, and storing the media off-site, if possible.
- Acquire and learn to use a good set of Boot/Root or emergency disks. Your choice as to which distribution to use may be influenced by whether or not it comes with a pre-built, comprehensive set of recovery disks.
- Don't be intimidated—you don't have to be a major guru or techie to get your work done. Following these simple guidelines should have you on your way to running a solid, reliable and stable SOHO Linux setup.
- Carefully weigh all options before investing time and money in software. One of the greatest benefits to using Linux is the freedom of choice—find the packages best for you.
- Be aware of how to share files and data with your non-Linux colleagues, clients and friends. Successfully interfacing with non-Linux users will make your life easier and may serve to win a few more Linux converts.

Well, now we know a little bit more about what we face if we choose to “go Linux” and leave our old OS behind. Next time, we'll move away from practical issues and discuss the future of Linux as it applies to the non-technical, SOHO end user. We'll also talk about how SOHO Linux users can get help from each other, without overwhelming the newsgroups and other traditional avenues of support. See you next time.

Resources

Norman M. Jacobowitz is a freelance writer and marketing consultant based in Seattle, Washington. Please send your comments, criticisms, suggestions and job offers to normj@aa.net.

Jim Hebert spends his spare time coming up with Stupid UNIX Tricks and dating the love of his life. He can be reached via e-mail at jhebert@compu-aid.com.

Archive Index Issue Table of Contents

Advanced search

SockMail

Noah Yasskin

Issue #53, September 1998

SockMail is described by its creators as a “100% Java client/server e-mail list management system”, and it is just that.

- Manufacturer: Sockem Software
- E-mail: info@sockem.com
- URL: http://www.sockem.com/
- Price: $295 US
- Reviewer: Noah Yasskin

E-mail saves time and money, particularly when you want to send multiple people the same message. One hundred letters can be sent as easily as one letter, and for the same price—absolutely nothing. Because of this, e-mail is the natural choice for distributing identical information to a group of people. Along with its benefits, though, e-mail has brought its share of problems—the biggest one is spam. However, just as there are legitimate uses for bulk mail, there are legitimate uses for mass e-mail mailings.

The Digital Mail Room

Smart companies on the Internet are replacing expensive paper mailings with digital mailings whenever possible. It looks antiquated when a company—especially one in the computer industry—sends me a press release by mail. Doing so reflects poorly on its understanding of technology. Physical goods

can't be delivered digitally, but information is destined to be transmitted like e-mail: effortlessly and inexpensively. However, many businesses are already having a difficult time managing the flow of e-mail and can easily have numerous mailing lists with thousands of addresses. Sockem Software's first product, SockMail, was created to help organizations effectively use lists of e-mail addresses. SockMail is described by its creators as a "100% Java client/server e-mail list management system", and it is just that. Web sites overburdened with extensive lists of e-mail addresses will find this application valuable.

Figure 1. SockMail Window

Sockem Software: A Java Start-up

Sockem Software is a 100% Java software start-up based in New York City's "Silicon Alley". The company is focused on Java because it believes the language's multi-platform and networking strengths will be successful in the marketplace. Java will provide the foundation for all of Sockem Software's client/server applications. In the company's eyes, a browser with a JVM (Java Virtual Machine) is the universal client. If Java is the mantra of Sockem Software, e-mail is the mantra of SockMail.

Sign Me Up

Web sites commonly have a text box in which people can sign up to be included on or dropped from a mailing list. This is usually done by having a CGI (common gateway interface) such as a Perl script send the information to someone who then manually adds and deletes names on a list of subscribers. It is difficult to automate this process with scripts and clumsy to paste names into a mail program in the blind copy field. SockMail's primary goal is to make the whole process of creating, maintaining and sending out information to lists of e-mail addresses as efficient as possible. It's a very simple and focused application that doesn't do much more than that. Users do a lot of the work by adding and deleting themselves from mailing lists. This is its big selling point for companies wanting to automate their web site subscription lists. SockMail's interface is functional but not pretty—it has the look of a shareware application.

Listserv and Majordomo

Unlike Listserv and Majordomo mail programs where users send commands within an e-mail to add or delete themselves from mailing lists, SockMail's lists are maintained through Java applets. Additionally, SockMail doesn't allow posting to people on a list. It is specifically designed for sending out mail to lists, not to facilitate communication between people on a list. A Listserv or

Majordomo system could be configured to have much the same functionality as SockMail, but these systems are more difficult to set up and maintain. SockMail is much easier to install and integrate with a web site or Intranet; however, it could never replace a Listserv or Majordomo program.

Figure 2. E-mail Window

Target Market, Target Mail

SockMail is designed for businesses with mailing lists of no more than 50,000 people. This number should be adequate for all but the largest company mailings and will hopefully dissuade spammers from abusing the application. It is an ideal tool for distributing press releases, e-zines, company updates, invitations or any targeted mailing. A large company could also manage internal mailing lists easily with SockMail. It is not designed for blind mass mailings or spamming. This product could be used to do that, but its potential to send out truly massive lists is limited. The lists have to be loaded into the browser, so available memory limits the potential number of recipients. Additionally, the server may run into memory problems sending to lists of over 50,000 names. In part, this is because Java uses static memory allocation for its applications. By default the server allocates around 20MB of memory, and this limit can be raised or lowered using options in the Java interpreter.

100% Java Installation

SockMail can be installed via a command line or GUI (graphical user interface). The application installs in literally seconds because the whole thing is only 197KB. It is designed to be purchased and downloaded over the Internet and is not available on CD-ROM or floppy disk. However, because SockMail is truly platform independent, it requires the use of the command line to start the GUI installation. This could be a little confusing for the novice Windows NT user, but a system administrator or anyone used to UNIX should have no problem getting the install up and running.

Figure 3. Preference Window

The install is done with JShield because the company insisted on keeping the application 100% Java; WinInstall uses Windows native code. However, it is not just a case of getting hooked on Java. Since it is 100% Java, the whole program can be installed remotely using TELNET. Most servers have remote administration capabilities, but taking its network-friendly cue from UNIX, everything on SockMail can done without being on a console. On Windows NT, this type of remote installation is rare, and Sockem Software has done a good job of building this functionality into the application.

The Server

To deliver client applets, the SockMail server must be running on the same computer as a web server, which limits its scalability. SockMail Pro, due out soon, will be able to run independently of a web server. The SockMail server has its own 100% Java database used to store e-mail addresses. It is a proprietary database without JDBC (Java database connectivity) functionality which could lead to problems in an enterprise setting. The server processes all requests from clients to add and delete names from lists and must have access to a local JVM (Java Virtual Machine). Although the server component is 100% Java, it can't function on most Windows 95 or Macintosh desktops, because most JVMs for Windows 95 and Macintosh don't support the TCP/IP socket connections the SockMail server needs. Security must be set up once the server is installed. Security features include an administrative login, IP filtering and separate administrative port. Only one user login is allowed. The server can handle multiple tasks, simultaneously updating one list and mailing to another.

Figure 4. Server Window

The Clients

SockMail uses three client applets: one to administer the server, one to add and one to delete addresses. Users can add and delete themselves from lists, but mailings can be performed only from the administrative applet; this applet is just 85KB and loads quickly. The **add** and **delete** clients are less than 30KB each. The applets can be customized to fit the look and feel of a particular web site. The clients run on any browser with a Sun compatible JVM version 1.02 or later. In case users don't have a Java-enabled browser, SockMail includes two HTML forms with CGI scripts. These can also be used to add and delete addresses.

Built-in Intelligence

SockMail includes the ability to intelligently retrieve e-mail addresses from web sites. A spider can search an entire web site, collect any e-mail addresses and add them to a specified list. This feature could give the product a bad name; however, the spider can search only about 1000 pages before running out of memory. This limits its usefulness as a tool for spamming. For quickly scanning a single URL, however, the spider works well. It is a useful tool for collecting contact information. It only picks up addresses after the **mailto** attribute.

To be more web friendly, the spider identifies itself as a robot. Webmasters can put configuration files on their pages to inform the spider not to search the site. As these intelligent agents become common, sites not wanting to be searched are starting to deny access to them.

SockMail also includes an easy way to check the InterNIC database for information about an e-mail address. For any registered domain, SockMail can query the InterNIC to find out the technical, administrative and billing contact information.

The Potential for Linux and Java

Sockem Software is a strong supporter of Linux. The company moved its own web site off of Sun Solaris and is a complete Linux shop. SockMail was developed on Linux boxes using Emacs, the newest versions of which have Java editing tools. The biggest advantage to developing Java on MS Windows remains the visual editing tools that are available on that platform. However, many visual development programs use version 1.1. of the JVM, which can lead to incompatibilities with version 1.0 used by most older browsers. By developing on Linux, Java programmers are assured they will be developing 100% Java because there are no proprietary JVMs on Linux. Organizations such as Blackdown (http://www.blackdown.org/) ported the JVM to Linux, and standard distributions of Linux include a JVM.

Looking Forward

Sockem Software is planning a SockMail Pro version, due sometime soon. This will address some of the current product's shortcomings. One limitation of SockMail is that it has only one user login. SockMail Pro will allow multiple user accounts with varying levels of access. SockMail Pro will also have its own Java web server to eliminate the need to have it running on the same computer as a site's web server, and will allow people to use the product independently of a web site. The SockMail Pro server will be able to run on a dedicated computer. In addition to a 100% Java version of the server, Sockem Software plans to have a Windows NT version compiled natively. This is being done with Supercede (a Paul Allen company). This NT-only version will also be available with WinInstall, so that the whole installation can be done through a GUI.

The Future of E-mail

SockMail is targeted towards a growing market. E-mail is undoubtedly the most popular application on the Internet. As much as the graphical WWW, it has fueled the astronomical growth of the Internet. E-mail is cutting down the amount of time we spend on the telephone, and may eventually force the handwritten letter to take its place in history next to the carrier pigeon and telegram. Like any other form of communication, though, e-mail requires courtesy and common sense. Nothing is as impersonal as a form letter, except possibly a form e-mail. This lesson applies to SockMail. A potential risk is involved when increasing the amount of e-mail we receive and send. Communication loses its value when it doesn't understand its audience.

SockMail allows individuals to easily sign up for mailings they're interested in and drop mailings they don't want. In this way, control over information is as much in the hands of consumers as producers. Let's hope this relationship stays in balance.

Noah Yasskin is a freelance writer living in Brooklyn, New York. He has a degree in History and Social Sciences from Eugene Lang College and attended the Philosophy program at the Graduate Faculty of Political and Social Science for a short while. Instead of Aristotle and Max Weber, he now writes about New Media companies and software programs. He can be reached at nyjup@aol.com.

Archive Index Issue Table of Contents

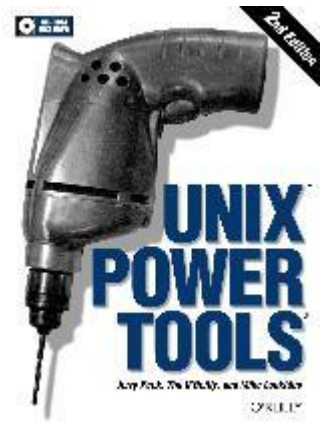
Advanced search

UNIX Power Tools

Samuel Ockman

Issue #53, September 1998

Using the shell interactively is becoming a bit of a black art.



- Authors: Jerry Peek, Tim O'Reilly and Mike Loukides
- Publisher: O'Reilly & Associates
- E-mail: info@oreilly.com
- URL: http://www.oreilly.com/
- Price: $59.95 US
- ISBN: 1-56592-260-3
- Reviewer: Samuel Ockman

The second edition of *UNIX Power Tools* is an impressive book by any measure. Numbering a hefty 1073 pages, it covers shells, editors and tools such as AWK, sed and RCS. The primary authors are Jerry Peek, Tim O'Reilly and Mike Loukides, but the book is made up of hundreds of individual articles by many different people, including UNIX luminaries Tom Christiansen and Simson Garfinkel.

UNIX Power Tools supplies many hints on how to save typing time by teaching you how to better use your shell interactively, how to take advantage of your editor and how to program scripts.

Using the shell interactively is becoming a bit of a black art. Luckily, *UNIX Power Tools* covers everything you need to know about this subject. Here are a few examples from the book to give you an idea of what it covers. First, here is how to edit three existing files: afile, bfile and cfile:

```
emacs [a-c]file
```

Many people already know how to do this; however, it works only if the files already exist. Lesser known is what to do if the files don't already exist. In this case, you edit all three at once by using:

```
emacs {a,b,c}file
```

With this information, it's easy to figure out how to make backup files:

```
cp filename{,.bak}
```

This command will copy the filename to filename.bak. Here's how to use this same idea to print six files or more:

```
lpr /usr3/hannah/training/{ed,vi,mail}/lab.{ms,out}
```

Are you confused by the **find** command? You won't have any problems after reading Chapter 17, which has 24 pages devoted to find. File permissions and processes are also covered in depth in their own chapters. Even wild cards get their own chapter. Whole chapters are also devoted to printing, terminal and serial line settings. (The book includes a great explanation of termcap and terminfo.)

vi is given two large chapters, while Emacs is relegated to being “the other editor” and given a scant ten pages. The vi coverage probably includes everything you'll ever need to know about it, while the Emacs section basically covers a few timesavers and the most essential commands, such as:

```
ESC-x psychoanalyze-pinhead
```

Missing from the book, though, is my favorite Emacs amusement:

```
ESC-9 ESC-x hanoi
```

The important thing to keep in mind is no matter what level of UNIX or Linux expertise you have now, you can learn much more. Some of the hints are beginner-oriented, but some may have even the most experienced user saying, “Hey, I never knew that...”

Here's a question for you. How do you run a program, while routing the standard error through a pipe to the mail program and leaving standard output on your screen under a Bourne shell? The answer is simple:

```
(program 3>&1 1>&2 2>&3 3>&-) | mail ockman &
```

Although bash and tcsh get special mention (as well as the more generic Bourne shell family and csh), the most powerful shell, zsh, is left out. Since so much of the book is devoted to shells, this is bothersome. Anyone who is really interested in having the most “power” will choose zsh. Still, since zsh is largely a superset of the other shells, almost all of the tips are still helpful.

sed is covered fairly extensively, and **awk** is covered sufficiently. Besides, in my opinion, all you need to know about sed and awk these days is that you use **s2p** and **a2p** respectively to translate your code to Perl. Speaking of Perl, the book has a few essays on why you should learn Perl, but offers no real help in doing so. It does offer the good recommendation of buying other O'Reilly books on the subject.

This book also does not cover networking or the X Window System, but that's good, because it leaves space for more important things. Nor, unfortunately, does it particularly make mention of Linux, although almost all information in the book holds true for Linux.

The book includes a CD-ROM of programs. Many of these programs are probably already installed on your Linux system, but you'll find quite a few others that will be useful. All of the programs are discussed in the text of the book. The CD includes source code and binaries for both Intel Linux and a handful of UNIX platforms. It does not make it clear which license the various programs fall under.

UNIX Power Tools is highly recommended. It's yet another amazing book from O'Reilly. After digesting all one thousand pages, you will be a wizard on the command line.

Samuel Ockman owns Penguin Computing (http://www.penguincomputing.com/). He spends all his free time rattling on about how great zsh is. His forthcoming book Super Advanced Programming in the Linux Environment is in the very early planning stages. You can e-mail him at ockman@penguincomputing.com.

Archive Index Issue Table of Contents

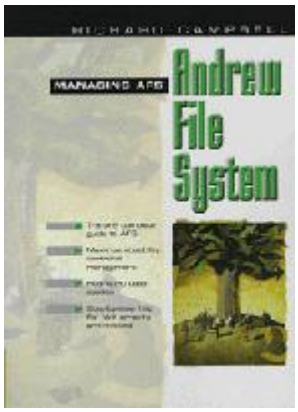
Advanced search

Managing AFS: Andrew File System

Daniel Lazenby

Issue #53, September 1998

From this book, the reader may gain an appreciation of the technical issues, skills and knowledge required to install, configure and manage an AFS environment.



- Author: Richard Campbell
- Publisher: Prentice Hall
- URL: http://www.phptr.com/
- Price: $45 US
- ISBN: 0-13-802729-3
- Reviewer: Daniel Lazenby

Managing AFS: Andrew File System provides a practical UNIX system administration view of the AFS file system. From this book, the reader may gain an appreciation of the technical issues, skills and knowledge required to install, configure and manage an AFS environment.

How This Book Can Help

Most vendor documentation focuses on how to install and configure the product and spends little time explaining the “why,” “when,” “where” or “what”

of the product. A third-party book is never meant to replace the vendor's documentation. Still, a third-party book can often fill in many of the gaps in the vendor's documentation. *Managing AFS* spends considerable time describing "why one would want to use AFS"; "what benefits can be derived from using AFS"; "where might one use AFS"; "what components comprise an AFS file system and the relationship of those components" and "what must be done to install, configure and manage an AFS Cell". Advanced AFS administration and how to debug an AFS installation are also addressed by Mr. Campbell.

The Book's Layout

Managing AFS is divided into 12 chapters and an appendix of AFS commands. The first two chapters provide an architectural and technical overview of AFS. Chapter 11 provides several AFS implementation case studies. A strategy and some tips for making a business case to support the use of AFS are provided in Chapter 12. The 50 or so AFS commands are briefly described in the Appendix. The sections in between the first two and last two chapters discuss setting up and managing an AFS Cell.

Chapters 3, 4 and 5 provide an introduction to AFS. These chapters cover setting up an AFS server, performing AFS operations on volumes and files, and setting up and administering an AFS client platform.

The focus of Chapters 6 and 7 shifts from system administration to AFS user account administration and security. Chapter 6, "Managing Users", describes how to establish AFS user accounts using Transarc's implementation of Kerberos. Administration of Transarc's Kerberos database is also discussed. AFS user login, authentication, groups and directory/file access controls are addressed in Chapter 7, "Using AFS". Transarc includes their implementation of some conventional UNIX user commands, programming commands and programs with AFS. Examples of UNIX commands and programs that have been modified by Transarc include chmod, df, close, lockf, ftpd, login and inetd. Differences between the two implementations are described.

Chapter 8, "Archiving Data", provides a momentary break from the other AFS administration concepts and tasks. As stated earlier, AFS supports the global distribution of files. With global distribution comes the challenge of file restoration. In addition to the user's data, data describing the AFS implementation and configuration must also be backed up. Challenges, tools and strategies used to back up and restore an AFS file system are presented here.

With the basics behind you, Chapter 9, "More AFS Administration", explores the finer details of AFS administration. Server management, updating AFS binaries,

job notification, changing the cell name, adding and removing database servers, adding and removing file servers, multi-homed servers and NFS-AFS gateways are just a few of the topics discussed.

Even a well-designed and implemented product will have problems. Chapter 10, “Debugging Problems”, offers a set of strategies for debugging an AFS installation. An explanation about when and how to use the available debugging tools is provided. This chapter also offers a set of typical AFS administration tasks that should be regularly performed and tested.

Why Consider AFS?

The original Andrew File System was created by a group of researchers at Carnegie Mellon University (CMU). They were striving to overcome the challenges associated with providing centralized file services in a distributed environment. Their AFS solution worked so well that many of the original researchers left CMU and formed the Transarc Corporation. AFS is now a registered trademark used by the Transarc Corporation to identify the commercial packaging of the Andrew File System. The AFS model was used as the basis for the Open Software Foundation's (OSF) Distributed File System (DFS) specification. Transarc has ensured there is a migration path from AFS to DFS.

A small shop with few workstations or shared files may have little need for AFS, whereas a large shop with many workstations, servers and the need to globally share files may have a greater need for it. In addition to being able to manage AFS servers and clients from a single workstation, AFS reportedly provides several other performance and financial benefits.

The book refers to “published” data on how AFS can support five to ten times more end users per server than other file systems. This increased user-to-server ratio translates into a need for fewer servers and fewer file storage administrators. An AFS file system can be made highly available using two or more AFS servers. This means that the loss of a server will not translate into a user being denied access to the file system.

One set of tests cited for an organization using NFS file sharing found that switching to AFS resulted in several performance improvements. For the same NFS type of workload, AFS resulted in a 60% decrease in network traffic. The server's load was decreased by 80%, and task execution time was reduced by 30%.

Linux AFS Port Availability

Transarc has ported AFS to most commercial UNIX platforms, as well as NT. Massachusetts Institute of Technology (MIT) has made several non-Transarc supported ports of AFS to various other architectures. The MIT ports used source code owned by Transarc. (The source code is reported to be available for a reasonable price.) Therefore, access to the MIT ports requires one to be an AFS licensee or affiliated with an organization who is an AFS licensee. Linux AFS is one of the several ports made by MIT.

Resources

Daniel Lazenby holds a BS in Decision Sciences. He first encountered UNIX in 1983 and discovered Linux in 1994. Today he provides engineering support for a range of platforms running Linux, AIX and HP/UX. He can be reached at dlazenby@ix.netcom.com.

Archive Index Issue Table of Contents

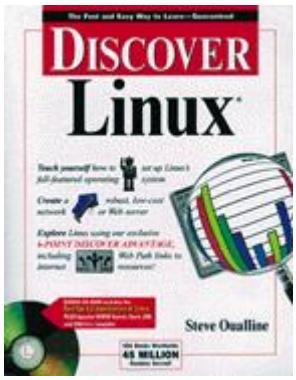
Advanced search

Discover Linux

Marjorie Richardson

Issue #53, September 1998

Not just another book for the total novice, Discover Linux is written for the Linux newcomer who is an experienced UNIX user.



- Author: Steve Oualline
- Publisher: IDG Books Worldwide, Inc.
- URL: http://www.idgbooks.com/
- Price: $25 US
- ISBN: 0-7645-3105-0
- Reviewer: Marjorie Richardson

Discover Linux is a good book. Mr. Oualline has a breezy style that is easy to read and understand. Indeed, the format of the book is designed for easy readability. The print is large, the margins wide, and there is plenty of white space to keep the eye moving to the next line.

Not just another book for the total novice, *Discover Linux* is written for the Linux newcomer who is an experienced UNIX user. It covers a lot of territory from the requisite installing Linux to setting up PPP to using Linux in the office. Many of the applications discussed are introduced with one or two short paragraphs, and a web address for obtaining more information. The word processor LyX and **ispell** are two that are handled this way. Other applications,

such as Applixware and **xv**, are given more space and have some options explained. Plenty of screen shots, all in black and white, go along with the text.

On the subject of screen shots, about midway through the book is a chapter (Part II, Chapter 8) on games in which every screen saver available with X is shown in black and white. To me, this seemed a bit of a waste of space. Also, all of the games are given just an introductory sentence or two and a screen shot. There is nothing in-depth here and that's fine. It is certainly not the purpose of this book to provide a tutorial on every application; rather, to give an overview that allows the user to pick from the many options which appeal to her.

Sprinkled throughout the book are tips, cautions and short anecdotes about various subjects; some are funny, some are not. All provide some information and a nice break.

The first part of the book is called “Up and Running with Linux”. It covers installation, getting started, getting help, configuration and backing up. The information provided is complete for those who already have some idea of what they are doing. The instructions for installing Linux are given in an easy-to-follow list format that should be understandable even to rank newbies. Just follow the steps—one, two, three—and you have a working Linux system. The book comes with the requisite CD-ROM which contains Red Hat Linux 4.2, a step behind, as usual with books, but certainly a very stable version for a newcomer. At any rate, the instructions for installation are therefore geared towards Red Hat users, as is true for all instructions throughout the book. Since Red Hat is the only distribution on the CD-ROM, this makes sense.

The second part of the book, “Fun and Games”, covers connecting to the Internet as well as the games mentioned previously. Again, Mr. Oualline provides step-by-step instructions for connecting with PPP, including discussions of **chat** scripts, **minicom** and **pppd**. He also tells you how to download and set up Netscape Navigator.

Part III is called “Linux in the Office” and covers mail programs, the X Window File Manager, Linux as a server and DOS. Mail programs, office applications and databases are treated similarly to the games. A short introduction is given with an example or screen shot and then a pointer is given to places on the Web for more information. The author presents Applixware as a bonus, stating “Move over Microsoft.” He ignores the fact that StarOffice now comes with Caldera, stating only that the German documentation makes it hard for English-only users. For Linux as a server, he does a good job of presenting SMB, NFS and NIS, telling you how to set up and use them. AMD, RAID, IPX and Appletalk are mentioned briefly.

Part IV, “Multimedia and Programming Tools”, follows the same pattern. Some subjects are given more attention than others, but you find out something about everything. An interesting chapter in this section is Chapter 14, “Understanding the Initialization Process”, which explains the boot process, including Run Levels and rc scripts.

Part V, “Advanced Configuration”, gives some details on configuring XFree86 and customizing your window manager. It also tells you what options are available if you get into trouble and how to report bugs.

Besides the normal Appendices, there is a section (all on blue paper, so you can find it easily) called “Discovery Center” which provides a quick reference for accomplishing tasks and points to the page in the book where these tasks are discussed more fully. I felt this section was a good addition, providing a quick way for the user to find the particular task she is interested in at the moment.

All in all, *Discover Linux* is a good reference for those with UNIX experience who are looking into this radical operating system called Linux.

Marjorie Richardson is Editor of *Linux Journal* and the e-zine *Linux Gazette*. She had been a programmer in the oil industry for 20 years before coming to SSC. She likes to quilt, read science fiction, watch action movies and musicals, go to the opera and motorcycle with her husband, Riley. She can be reached via e-mail at info@linuxjournal.com.com.

Archive Index Issue Table of Contents

Advanced search

Updating Pages Automatically

Reuven M. Lerner

Issue #53, September 1998

Have a need to change a file on your web site on a daily or monthly basis? This month Mr. Lerner tells us how to do it.

The home page of my web browser is set to http://www.dilbert.com/, home of the famous and funny Dilbert comic strip. Thanks to the magic of the Internet, I'm able to enjoy Dilbert's tragicomic humor each morning, just before I start my workday.

The Dilbert web site would not be very useful or interesting were it not for the creative talents of Scott Adams, Dilbert's creator. What makes it interesting from a technical perspective is the way in which the comic is updated automatically each day. Every morning, the latest comic is automatically placed on the Dilbert home page, giving millions of fans the chance to see the latest installment.

This month, we will examine several ways in which you can create pages that are automatically updated, so that a user can discover new content at the same URL each day. We will look at several different means to the same end, ranging from CGI programs to **cron** jobs, and will even take a brief look at how to use databases when publishing new content.

Pointing with CGI

For starters, let's assume our web site consists of seven different pages, one for each day of the week (e.g., file-0.html on Sunday, through file-6.html on Saturday). How can we configure the site so that people requesting today.html (or today.pl) will be shown today's file? In other words, a visitor on Wednesday should be shown file-3.html when requesting today.html. Such a system might be appropriate for a school cafeteria, where the food tends to be the same each day of the week.

Perhaps the simplest solution is a CGI program, which we will call today.pl. If we write the program in Perl, we can easily determine the day of the week using the **localtime** function, which returns a list of elements describing the current date and time. Using the sixth element of that list, which indicates the current day of the week, we can create the correct URL for that day. Finally, we can use the HTTP “Location” header to redirect the user's browser to the correct location.

A simple implementation of this program is shown in Listing 1. The program should seem familiar to anyone who has written CGI programs. It enables all of Perl's warning systems: **-w** for optional warnings, **-T** for extra security, **strict** for extra compile-time checking and **diagnostics** for more complete documentation if something fails.

By using CGI.pm, the standard Perl module for writing CGI programs, we gain easy access to any input passed by the server, as well as the various output methods a CGI program might use. Most CGI programs use the output methods meant for returning HTML to a user's browser, including sending a MIME “Content-type” header indicating the type of content about to be sent—in our case, we return a “Location” header, which removes the need for a “Content-type” header.

If the above program is installed as /cgi-bin/today.pl on our server, visitors will always be greeted with the appropriate file for the current day of the week.

The above program, simple as it is, has several flaws. Most significantly, CGI is slow and inefficient; using it to redirect the user's browser to another file will slow down the user's experience, as well as increase the load on your server. Each time a CGI program is invoked, the server must create a new process. If the program is written in Perl, this means the Perl binary must be started, which can take some time.

One solution might be to use **mod_perl**, which inserts a fully working version of Perl into the Apache web server. Using mod_perl means Apache no longer needs to create a new process, execute the Perl binary or compile the Perl program, which will cut down on server resource use. However, this still means that each time a user requests the home page, the server must execute a program. If the page is requested 1,000 times in a given day, then the program will run 1,000 times. This might not sound like much, but imagine what happens when your site grows in popularity, getting 1,000,000 hits each day.

Even this solution doesn't address the fact that not all users run browsers which handle redirection. If a browser does not handle the notice, the user will

be unable to see today's file. This problem is increasingly rare, but keep it in mind if you want the maximum possible audience for your web site.

Automatically Copying Pages with cron

Let's now examine a strategy in which the program runs only once per day, regardless of how many people ask to see today's page. This method reduces the load on the server and allows people with old browsers to visit our site without any trouble. The easiest strategy is to use Linux's **cron** utility, which allows us to automatically run programs at any time. Using cron, we can run our program once per day, copying the appropriate file to today.html. On Sundays, file-0.html will be copied to today.html, while on Thursdays, file-4.html will be copied to today.html.

Listing 2 is an example of such a program. If this program were run once a day, then today.html would always contain the file for the appropriate day. Moreover, the server would be able to respond to the document request without having to create a new CGI process or use Perl.

The above program is *not* run through CGI, but rather through cron. In order to run a program through cron, you must add an entry to your **crontab**, a specially formatted text file that describes when a program should be run. Each user has a separate crontab file; that is, each user can arrange for different cron jobs to run at different dates and times.

You can edit the crontab file using the crontab program, which is typically in /usr/bin/crontab. To modify your crontab file, use **crontab -e**, which brings up the editor defined in the **EDITOR** environment variable. The format of crontab is too involved for me to explain here; typing **man 5 crontab** on the Linux command line will bring up the manual page describing the format. (Typing only **man crontab** will bring up a description of the crontab program, rather than the crontab file format, a distinction which can be confusing to new users.)

Assuming we want to run the above program (which I have called cron-today.pl) at one minute after midnight, we could add the following entry to our crontab:

```
1 0 * * * /usr/local/bin/cron-today.pl
```

In other words, we want to run /usr/local/bin/cron-today.pl at one minute after midnight (**1 0**), every day of the month (*****), every month (*****), and every day of the week (*****).

The output from each cron is e-mailed to the user who owns that job. After installing the above line in my crontab, I receive e-mail from the cron job each

day at approximately 12:01 a.m. And each day, anyone visiting our site was shown the correct file for today.html.

Using Symbolic Links

The above cron-based technique works, but has some annoying side effects. For example, what happens if you decide to change the Tuesday menu on Tuesday morning? The change will not be reflected until the following Tuesday, because today.html contains the contents of file-2.html from 12:01 a.m., when the snapshot was taken.

In order to solve this problem, as well as reduce the disk space used by two copies of the program, we can use symbolic links. These look like files, but are really pointers to files, similar to Macintosh “aliases” or Windows “shortcuts”. If we create a symbolic link from today.html to file-0.html, the two file names will be equivalent for most purposes. (Other “hard” links are also available under Linux, but are more limited.)

If we want to create a symbolic link named today.html that points to file-0.html, we say

```
ln -s file-0.html today.html
```

If you want to change the link so that it points to file-1.html, remove the old link and create a new one, like this:

```
rm -fv today.html
ln -s file-1.html today.html
```

Alternatively, we can use the **-f** (“force”) option to **ln**, forcing the link assignment even if it was previously linked elsewhere:

```
ln -sf file-0.html today.html
```

If we were to do this each day, removing the old link and creating a new one, we would be doing effectively the same thing as in cron-today.pl, but with the added advantage of equating the two files. In addition, we would be saving space on the file system by pointing to the original file rather than copying it.

Listing 3 contains a short Perl program meant to be run via cron, which creates such a link. Anything sent to standard output (STDOUT) via “print” statements is sent to the owner of the cron job. This program assumes the owner of the cron job (under whose user ID the program is run) has permission to remove the existing file, as well as create a new symbolic link in the directory. It is possible to create a symbolic link to any file, including a nonexistent file; only when you try to access the file are the permissions checked.

Publishing Daily Items

The techniques we have examined so far are most useful when the same item appears each week or perhaps each month. In many cases, though, publishing on the Web involves creating a new file each day and making that available. For starters, we will look into how to create a new file each day (of the form file-1.html, as before), so that the newest file will be available by looking at today.html.

Once again, we could accomplish this with either a CGI program or a cron job, examples of which you can see in Listing 4 and Listing 5, respectively. Both programs use the same basic algorithm to find the highest-numbered file of the form file-n.html, where n is the sequential number for the file.

The key to both programs is in these lines:

```
if (opendir(DIR, $directory))
{
@files = sort by_number
   grep {/^file-[0-9]+\.html$/} readdir(DIR);
closedir DIR;
}
```

First, we open **$directory**, the directory in which the files exist. (If the program cannot open the directory, it logs an error.) We then read the contents of the directory DIR, using Perl's **grep** function to filter out any files not fitting the file-n.html pattern. Finally, we sort those files with our own **by_number** routine, which compares the sequential numbers rather than the full file name.

Once we have the list of files, we pick off the last element of @files, which has the highest sequential number. We can then redirect the user's browser to that file using CGI.pm's **redirect** method.

If we want to publish items each day, we should try a better system than this one, which depends on sequential numbers. First of all, it is easier to handle file names which mention the subject (e.g., menu.html) or the date (e.g., file-1998-06-01), rather than something named with sequential numbers, as in file-3023.html.

Secondly, arranging articles by date provides users with a natural way of navigating through archives in the future without having to depend on the site's navigation scheme. In addition, creating file names according to date rather than sequential numbers decreases the chances of error.

If you choose to use the date in the file name, as in file-1998-06-01, try to keep the date elements in year-month-day order, so that sorting file names alphanumerically will also sort them chronologically. Then, we can write a small

program to select the file for today based on the date and run it each day with cron. An example is shown in Listing 6. The program logic is fairly straightforward, taking the date information from our call to localtime and piecing those elements together to create the file name.

However, problems may arise if the file for today does not exist. As I mentioned earlier, symbolic links do not have to point to files; they may point to any valid file name, even if no file by that name exists. However, if the symbolic link points to a non-existent file, users will be greeted with a dreaded "404--File not found" error upon loading today.html from our site. A more sophisticated version of this program would check to see if a file corresponding to today's date existed on the site. Such a program would then search backward (or forward, if you prefer) chronologically to find the best match for the today.html symbolic link. It could even send e-mail to the webmaster indicating that such a problem existed.

Using Databases

One additional method for publishing material on the Internet regularly is using databases. Rather than relying on file names keyed with particular dates, we can create a table that establishes a correspondence between file names and dates. We can then write a CGI program to retrieve the current file or a program meant to be run via cron to create a symbolic link to the current file.

Another option is to store the files inside of the database. However, if we were to do that, we would also have to make it possible for the site's editors and designers to store, retrieve and edit the information inside the database. For our purposes, we will assume the files exist on the server's file system, and we are trying to point to them rather than store their contents in a different way. These examples were tested under Red Hat 5.1, Perl 5.004_04, the database interface (DBI) libraries for Perl, and MySQL, a mostly free relational database system available from http://www.mysql.com/.

Before we can do anything else, we have to create a table to hold the information. The table will be relatively simple, containing only file names and dates. We will assume that each article can be published on only one date, but that each date can contain multiple articles, which makes our table creation command look like the following:

```
CREATE TABLE Articles
 (filename  VARCHAR(100) NOT NULL PRIMARY KEY,
 date    DATE NOT NULL);
```

In the above, we define **filename** as a 100-character text field, which must be filled in (**NOT NULL**) and cannot be the same as any other file name (**PRIMARY KEY**). If we try to insert the same file name on two different dates, the database

will stop us. By contrast, because we want to allow more than one file on a given date, the **date** field (which has a type of **DATE**) is defined as **NOT NULL**, meaning that we must indicate a date with each file name.

In order to add a file to our database, we can use the following SQL command:

```
INSERT INTO Articles (filename, date)
VALUES ("foobar.html", "1998-06-05");
```

If you are using MySQL, you must put quotation marks around the date, or the default date of **0000-00-00** will be inserted.

In addition to the confirmation message (**1 row affected**) we receive upon submitting the above query, we can check the contents of the table:

```
mysql> SELECT * FROM Articles;
+-------------+------------+
| filename    | date       |
+-------------+------------+
| foobar.html | 1998-06-05 |
+-------------+------------+
1 row in set (0.08 sec)
```

Entering information into a database using raw SQL is inefficient, prone to errors and unhelpful for users who are unfamiliar or encomfortable with SQL. Listing 7 contains an HTML form that can be used to enter new articles into the database, using the program in Listing 8.

Finally, we will need a version of today.pl that retrieves the file for today. A CGI version of the program is in Listing 9; rewriting it such that it uses cron should be fairly straightforward. A more sophisticated version of the program would even check to see if the named file exists, searching backward.

Publishing regular articles on the Web is far less complicated than publishing a daily or weekly newspaper, but still involves a bit of planning and programming. In addition, no matter what method you choose, you will still have to make some trade-offs between performance and flexibility. Nevertheless, creating a page that changes each day and provides access to the site's archives is not especially difficult and can provide enough variety to draw people.

All listings referred to in this article are available by anonymous download in the file ftp://ftp.linuxjournal.com/pub/lj/listings/issue53/3060.tgz.

Reuven M. Lerner (reuven@netvision.net.il) s 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.

Archive Index Issue Table of Contents

Advanced search

Letters to the Editor

Various

Issue #53, September 1998

Readers sound off.

BTS Issue 50

In the article on ESDI drives, the URL for the MCA page should be: http://glycerine.cetmm.uni.edu/mca/. I love your magazine. Three articles in this issue answered questions I had been having. Keep up the good work.

—Pete dstrader@zianet.com

Linux and Compaq ProLiant 2000 and SMART

I have read articles saying many wonderful things about Linux and I believe most of them to be true. Unfortunately, the extent of hardware support that some of these articles claim is not a reality—or at least does not seem to be when it comes to Compaq equipment.

I have just spent the best part of a day searching the Internet by various means, including various search engines, trying to find drivers to support the embedded NCR 53C710-based SCSI controller in a Compaq ProLiant 2000 and also drivers to support a Compaq SMART SCSI RAID array controller. Result: nothing, except a lot of stress.

Please can someone help me (and the many others who I have encountered looking for these drivers). Linux claims to support quite a bit of hardware—please extend this support to include some key Compaq server items.

—Graeme Nelson graeme@cheerful.com

Open Source vs. Free Software

I'm writing you after seeing one too many odes to the glories of the Open Source movement. I have a serious problem with the whole Open Source bandwagon due to the fact that Open Source is almost solely about making free software palatable to business—a segment of society which has played a largely non-existent role in the development of free software. Business has done nothing to make the user and programmer community at large more aware of the benefits of free software. I feel the primary benefits are individual and social freedom.

The June article by Eric S. Raymond, “Open Source Summit”, is a good example of the fundamental emptiness of the Open Source movement. The O'Reilly conference report struck me as being more about how Larry Wall, et al., can strike it rich than about how the lives of users and programmers can be enhanced through free software. I have nothing against people being financially compensated for their labor, but being financially compensated for one's labor has always been a secondary or even irrelevant consideration in the free software movement and rightfully so.

The most appalling notion implied in the rhetoric of the Open Source movement is that we, those of us who use/write/support free software, have to change our ways and adopt a more corporate mindset if we want free software to be successful in the real world. This is manifestly ridiculous. If free software hadn't already proven itself thoroughly in the real world, there wouldn't even be an Open Source movement. In fact, I think that free software and the free software movement have proven themselves to such an amazing degree that the corporate world now wants to find a way to squeeze a buck out of us. Again, there is nothing wrong with making a buck, but don't you dare do it at the expense of my freedom.

Unfortunately, free software developers are not a major source of advertising dollars for *LJ*, so it is not likely that *LJ* will be publishing alternate views to the Open Source camp anytime soon. That apparently being the case, I would suggest that if *LJ* readers are interested in an alternate view of the free software movement, check out, for starters, Richard Stallman's article “Why Free Software is better than Open Source” at http://www.gnu.org/philosophy/open-source-or-free.html.

—Shawn Ewald shawn@wilshire.net

While I disagree with your stated beliefs, I'm always happy to publish alternate views—I have done so in the past, do so now with your letter and will do so again in the future. While it is true that *LJ* does not receive advertising dollars

from free software, we put free software items in the “New Products” column and publish reviews and tutorials of free software.

Linux Journal strongly supports “freely available” software and the Open Source movement. This is one reason we chose the Debian distribution to use in our office.

By the way, I see no reason for you to have singled out Larry Wall as looking for a way to “strike it rich”. Perl is free and Larry is most definitely not a money-grubbing type of guy.

—Editor

PPPui

My thanks to the numerous people who've written in response to my article in *LJ* #50, “PPPui: A Friendly GUI For PPP”. To anyone interested in more features —especially anyone who relies on single-use passwords—please check http://www.teleport.com/~nmeyers/PPPui/ for features added to PPPui since the article was originally submitted.

—Nathan Meyers nmeyers@teleport.com

Sybase and Linux

I just wanted to mention that we have a couple of people in our lab who provide Sybase connectivity (server running on Irix) in their Linux programs. They told me that it is fairly easy to get it working using freely available C code downloaded from the Web.

—Marjan Trutschlmtrutsch@cs.uml.edu

PPPui Alternative

In Issue #50's article, “PPPui: A Friendly GUI for PPP”, Mr. Meyers notes that PPP does not have a good user interface: the only way you know if your connection succeeded or failed is to check the process list. Mr. Meyers offers a solution.

There's actually a simpler way than his program: direct syslog to Console 9 as described in an earlier issue of *Linux Journal*, and enable logging in **chat** using **--v**. Then, just hit **alt—f9** to view your syslog console, and you can watch the progress of chat's attempt to connect. Failed connects show up as Alarm, exit or hangups; a sluggish connection can be observed as **pppd** sends **EchoReq**'s out. Disconnects show as hangups. This live syslog is also invaluable when debugging your chat script.

—Cynthia Higginbothamcyhiggin@pipeline.com

Caldera Review—Not!

So there I was, reading the review of Caldera OpenLinux (June 1998), and the reviewer, Sid Wentworth, writes:

> Caldera, by default, uses the Looking Glass Desktop. Not being a desktop sort of guy, I am not particularly excited about it, but, if you want a desktop, it seems adequate.

Great! He's not a desktop sort of guy. Why is a “non-desktop” kind of guy getting paid for reviewing anything other than AWK scripts?

It's hard to take seriously reviews that completely leave out subjective comments about functionality a reviewer doesn't really have an interest in, or more to the point, his audience does have an interest in. Don't care about it, don't review it—simple! The rest of us, though, might have been interested in the state of this product's constantly evolving user environment, but the heck with us. You need better writers, which shouldn't be too difficult. At least the Windows techies take apart the toys they review.

—Tim Parsonstsparsons@earthlink.net

There is a lot more to Caldera OpenLinux (or any Linux distribution) than the desktop. Also, desktop choices are available with any Linux flavor. In the case of Caldera OpenLinux, I found their proprietary desktop to be unexciting, but I also found that to be unimportant over all.

I could have easily written a 50-page review of this product—so many capabilities are there to discuss. For example, each language translator could have been reviewed. Even if I just looked at GUI capabilities, a comprehensive review of desktops would need to include comparisons with XFM and other free file managers.

Don't get me wrong. If Looking Glass had been exciting, I would have talked more about it. It wasn't and I don't think it matters. What did matter, as I said in the article, was StarOffice which, by the way, is its own desktop.

—Sid info@linuxjournal.com

Summit Article

Fabulous! I wish I'd been there!

—Tim O'Reilly, O'Reilly & Associatestim@ora.com

Small Error

I believe that the “Best of Tech Support” column in the May 1998 *LJ* contains a small error. Regarding the question of Linux's behaviour when a file system is infected with an MS-DOS virus, Chad Robinson states that because Linux spreads file system meta-data more evenly throughout the physical disc, “random potshots” are more likely to cause corruption of the meta-data. This is misleading, in that if the amounts of meta-data were the same, the probability of a potshot hitting the part of an MS-DOS file system containing meta-data is equal to that of it hitting more evenly distributed meta-data on an EXT2 file system. “Concentration” does not affect the probability, only the amount of meta-data.

—Sidney Cammeresicammeres@uiuc.edu

Concerning LTE Issue 50

Simon Maurice in the June 1998 *LJ* criticizes Red Hat v5.0 for perceived shortcomings. Is he truly serious in saying that Red Hat's RPMS is a “Microsoft-like effort”? I think the package management system is one of the features that puts Red Hat distributions at or near the top of the pile. Where are all the bugs he alleges? I've found a few wrinkles, but nothing I'd call a serious bug.

As to the alleged problems with the actual distribution, I have run both the v4.2 and the v5.0 distributions as “official/supported” releases. I haven't applied any patches—yet my system is so stable it hasn't crashed since March 1998 when RH5.0 was installed. (Which is more than I can say for my Windows NT4 system.)

However, I do agree that Red Hat should be more polite when it comes to customer (e-mail) support. I asked several questions about the v4.2 distribution and was curtly told the questions were outside the support structure and to try the mailing lists. I wasn't happy with that answer, but it taught me to go to the documented sources first.

For $50 or so, Red Hat's distribution is by many orders of magnitude easier to use and install than the first distribution of Linux I bought back in 1993 (Trans Ameritech, v0.90 kernel). Compared to commercial operating systems (e.g., SCO, Windows NT), there is no comparison, whether for value for money, stability or user support. Where else but in the Linux community can you get bug fixes (if they are needed) so quickly?

—Alan Nutleyanutley@halenet.com.au

BTS Comment

I just received the June 1998 *LJ* (love> that airmail delivery folks). In “Best of Technical Support”, Chad Robinson and Pierre Ficheux both gave useful answers to “how do I back up NT and Linux”. However, you should know that the AMANDA package (Advanced Maryland Automated Network Disk Archiver, http://www.amanda.org/) works well with NT as well as with just about any flavour of UNIX. The current release version is 2.4.0 (with 2.4.1 in snapshot development form), and people all over the world are using it to back up all sorts of machines.

—James C. McPhersonjcm@kalessin.humbug.org.au

Archive Index Issue Table of Contents

Advanced search

How Many Distributions?

Marjorie Richardson

Issue #53, September 1998

I can't help but feel that the community would be better served if these enthusiasts picked their favorite distribution and contributed toward making it the ultimate distribution.

I have been seeing what I consider to be a disturbing trend in the Linux community. A computer wizard discovers Linux, decides it's cool, but doesn't think any of the current distributions are adequate. So, he gets together with a few of his friends and begins working on “yet another” Linux distribution. Don't get me wrong, these guys are great! They are investing a lot of time and effort in attempting to put together the “boss” distribution.

Somehow though, I can't help but feel that the community would be better served if these enthusiasts picked their favorite distribution and contributed toward making it the ultimate distribution. The more distributions are available, the more difficult it becomes for newcomers to make a choice and experts to keep up with them all. Even worse, as distributions become more and more divergent, Linux applications will not work on all of them. Thus, they are competitors with each other rather than united in competition with Microsoft and Apple.

In calling for a Linux Standard Base System Project, Bruce Perens said:

> Binary compatibility between Linux distributions has become a casualty of the competition between them. There are vast differences in versions of libraries, etc., that make it difficult for a commercial application to target more than one Linux distribution. This fragmentation is one of the main reasons that UNIX was crippled in the computer market.

Bruce is right: there needs to be a base standard. Then, using that standard, programmers should contribute to their favorite distribution rather than

creating a new one. Let's be sure that Linux doesn't follow the same path as UNIX.

In a similar vein, if the newcomer is not a programmer, he often chooses to create another resources web page rather than a distribution. Every week I hear of a new Linux page. Again, these pages have excellent material, but a lot of them duplicate each other. When another newbie looks for Linux information, he finds not just a few Linux pages, but many. Sorting through them all is time consuming and confusing. I think a better method for the new enthusiast would be to pick a page that's been around a while and appeals to him personally, then offer to contribute to it. Most webmasters for these pages are glad to receive current information.

SSC has always been willing to provide space for new sections and add information provided by others in order to ensure that our Linux Resources page is as comprehensive and attractive as possible. This attitude is not unique to us—slashdot.org and Linux Weekly News are other sites willing to accept contributions from the community.

I'm not saying we should have only one Linux distribution; we just don't need one for every new person who discovers Linux. The message here is, don't reinvent the wheel—pick your favorite distribution or resources page and help make it better.

Next Month

In our October issue, we have a great article about how Cisco Systems is using Linux print servers worldwide. The author, Damian Ivereigh, discusses technical issues involved with the print system and provides the method and code for solving common problems. If you work for a large corporation (or even a small one) with chaotic print services, this article is a must read.

As usual, I had more articles for our graphics focus issue than would fit in this magazine. So, in our next issue we will continue this focus with an article about a set of audio tools for Linux called Sculptor. These tools can be used for manipulating audio spectra and providing continuous audio output.

Outsourcing Subscriptions

Once again, *Linux Journal* has outsourced its subscription services; this time to a fulfillment house in Missouri City, Texas. We truly believe that in the long term, this solution will provide the best service to our subscribers. Unfortunately, this house was not prepared for the massive amount of e-mail *LJ* receives, and so is off to a shaky start. I apologize to those of you who got caught in this transition and did not receive timely answers to your mail. I

expect that by the time you read this, all problems will have been solved, and services will be running smoothly.

Do your part to help cut down on the amount of e-mail subscription services receives. Before writing complaint e-mail, check our web site for your subscription status and finger info@linuxjournal.com for the actual mailing date of the current issue.

Upcoming Events

- 6th USENIX Tcl/Tk Conference, September 14-18, 1998, San Diego, CA, http://www.usenix.org/events/tcl98/
- ISPCON Fall '98, September 28-October 1, 1998, San Jose, CA, http://www.ispcon.com/
- DECUS '98, October 3-8, 1998, Los Angeles, CA, http://www.decus.org/
- Atlanta Linux Showcase, Second Annual, October 23-24, 1998, Atlanta, GA, http://www.ale.org/showcase/

Archive Index Issue Table of Contents

Advanced search

USENIX 1998

Aaron Mauck

Issue #53, September 1998

SSC's system administrator travels to New Orleans and actually returns to tell us about it.

Each year, the USENIX organization (http://www.usenix.org/) puts on a technical conference dealing with UNIX and other UNIX-like systems. This year they had an emphasis on free or Open Source operating systems, primarily Linux and *BSD. The conference was held in New Orleans, Louisiana from June 15th to the 20th.

Tutorials

Many day-long tutorials were offered on Monday and Tuesday including “Inside the Linux Kernel” by Stephen Tweedie, one of the EXT2 developers, and several talks on Networking and Security. I attended “Hot Topics in System Administration”, given by Treni Hein and Evi Nemeth. They covered many topics including Samba, Packet Filtering and IPv6.

Linux Journal Booth

Vendor Expo

I found it refreshing to see a vendor exposition (albeit a small one) comprised completely of UNIX-friendly companies. O'Reilly was there, displaying all of their titles for sale at 20% off. Needless to say, this made it one of the most popular booths. Most of the faces were familiar: Red Hat, Linux International, InfoMagic, the three heads of BSD and others. Among the unexpected participants was the FBI, just a short distance from the Free Software Foundation. The whole atmosphere of the exposition was quite relaxed, without the hectic feel of COMDEX and other large industry trade shows.

The LI booth staff are happy to see us.

BOFs and Speeches

Each evening offered several talks by different people on a wide range of subjects. I caught “The State of Linux” talk by Linus Torvalds on Thursday afternoon. He set Aug/Sep 98 as a hopeful release date for the 2.2 kernel. Another event that took place every evening was the “Birds of a Feather” (BOF) meetings, which were designed as a place for people with common interests to come together and discuss their ideas and goals. It was also a great place to rub shoulders with some of the “big names” in the UNIX community, such as Keith Bolstic, Eric Allman and Jon “maddog” Hall.

Terminal Room

What UNIX conference would be complete without a terminal room? Luckily, Earthlink and openBSD donated machines and bandwidth and created a room with thirty or so machines running openBSD, connected to a T1.

The terminal room at full-tilt boogie.

Summary

If I were to do it all over again (and I most definitely want to), I would spend more time planning what I want to learn. I was a bit overwhelmed by the sheer number of talks/events, and therefore found it difficult to focus on exactly what I wanted to get from the experience—I was constantly spreading myself too thin. For any UNIX, Linux, BSD etc. lover, USENIX is a must at least once in a lifetime. It is a very friendly and co-operative environment and has definitely earned its reputation as one of the hubs of the computing community.

Jon “maddog” Hall and admirers

Aaron Mauck is the System Administrator for SSC. He can be reached via e-mail at info@linuxjournal.com.

Archive Index Issue Table of Contents

Advanced search

A Little Devil Called tr

Hans de Vreught

Issue #53, September 1998

Here's a useful command for translating or deleting characters in a file.

The program called **tr** is not a big program; it is quite small and not extremely powerful. However, if you write scripts, you will treasure it as one of your favorites. It is a typical script program, reading from stdin and writing to stdout; there are no file names to provide as arguments. The main function is translating characters. A second important function is deleting characters. Furthermore, tr is capable of squeezing repeated characters into one, but that particular function is rarely used.

Let us begin with translating characters. The tr command takes the form:

```
tr
```

While tr reads its input, it replaces characters appearing in string1 by the corresponding characters in string2. So, the command **tr abc def** will replace a line like “the quick brown fox quickly jumped over the lazy dog” into “the quifk erown fox quifkly jumped over the ldzy dog”. Well, that doesn't make sense, but it does demonstrate how tr works.

Have you ever wanted to capitalize or de-capitalize a file? To capitalize it, you can use:

```
tr abcdefghijklmnopqrstuvwxyz \
        ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Luckily, we can also use ranges of characters to specify the characters more efficiently:

```
tr a-z A-Z
```

Ever had those horrible upper case DOS file names? Here's a Bourne script to take care of them:

```
for f in *; do
      mv $f `echo $f | tr A-Z a-z`
   done
```

Many UNIX editors allow some text to be processed by the shell. For example, to replace all upper case characters of the next paragraph with lower case while in **vi**, type:

```
!}tr A-Z a-z
```

As another example, the command:

```
!jtr a-z A-Z
```

capitalizes the current and next line (the character after the **!** is a movement character).

If you read the International Obfuscated C Code Contest (ftp://ftp.uu.net./pub/ioccc/), you frequently see that part of the hints are coded by a method called **rot13**. **rot13** is a Caesar cypher, i.e., a cypher in which all letters are shifted some number of places. For example, a becomes b, b becomes c, ..., y becomes z, and z becomes a. In rot13 each letter is shifted 13 places. It is a weak cypher, and to decipher it, you can use rot13 again. You can also use tr to read the text in this way:

```
tr a-zA-Z n-za-mN-ZA-M
```

Another interesting way to use tr is to change files from Macintosh format to UNIX format. For returns, the Macintosh uses **\r** while UNIX uses **\n**. GNU tr allows you to use the C special characters, so type:

```
tr \r \n
```

If you don't have GNU's version of tr, you can always use the corresponding octal numbers as shown here:

```
tr \015 \012
```

You might wonder what would happen if the second string is shorter than the first string. POSIX says this is not allowed. System V says that only that portion of the first string is used that has a matching character in the second string. BSD and GNU pad the second string with its final character in order to match the length of the first string.

The reason this last method is handy becomes clearer when we take complements into account. Assume you wish to make a list of all words and keywords in your listing. When you use **-c**, tr complements the first string. In C, all identifiers and keywords consist of **a-zA-Z0-9_**, so those are the characters we want to keep. Thus, we can do the following:

```
tr -c a-zA-Z0-9_ \n
```

If we pipe the tr output through **sort -u**, we get our desired list. If we follow POSIX, the second string would have to describe 193 newline characters (described as **\n*193** or **\n***). If we use system V, only the zero byte is translated to a newline, since the complement of **a-zA-Z0-9_** starts with the zero byte.

The second important use of tr is to remove characters. For this option, you use the flag **-d** with one string as an argument. To fix up those nasty MS-DOS text files with a **^M** at the end of the line and a trailing **^Z**, specify tr in this way:

```
tr -d \015\032
```

Many people have written a program in C to do this same operation. Well, a C program isn't necessary—you only need to know the right program, tr, with the right flags. The **-d** flag isn't used often, but is nice to have when needed. You can combine it with the **-c** flag to delete everything except characters from the string you supplied as an argument.

Repeated characters can be squeezed into a single one using the **-s** option with one string as an argument. It can also be used to squeeze white space. To remove empty lines, type:

```
tr -s \n
```

The **-s** option can be used with two strings as arguments. In that case, tr first translates the text as if **-s** were not given and then tries to squeeze the characters in the second string. For instance, we can squeeze all standard white space to a single space by specifying:

```
tr -s   \n [ *]
```

The **-d** flag can also be used with two strings: the characters in the first string will be removed and the characters in the second string will be squeezed.

tr may not be a great program; however, it gets the job done. It is particularly useful in scripts using pipes and command substitutions (i.e., inside the back quotes). If you use tr often, you'll learn to appreciate its capabilities. Small is beautiful.

Hans de Vreught (J.P.M.deVreught@cs.tudelft.nl) is a computer science researcher at Delft University of Technology. He has been using UNIX since 1982 (Linux since 0.99.13). He likes non-virtual Belgian beer, and he is a real globetrotter, having already traveled twice around the world.

Archive Index Issue Table of Contents

Advanced search

Training on a Token Ring Network

Charles Kitsuki

Issue #53, September 1998

Linux can provide technical managers with cost-effective, reliable training tools

Money is not always available to do training in a business environment. Overall, businesses are looking for cost-effective solutions. Finding training tools that are both cost effective and reliable is not easy. This is especially true in a technical environment. To teach UNIX and web-based technology, Linux can provide technical managers with solutions to this dilemma. This article describes how to set up a Linux system for training on a token-ring network.

Convincing others of the benefit of using Linux as a training tool was not a problem in my particular scenario; this may not be the case in other environments—it depends on the audience. My situation involved introducing the product to a liberal management staff. Their main concerns were maintenance and material costs; they wanted a system that would not add substantial cost or bring additional work to other groups. Linux fulfills both criteria.

Completed projects provided the equipment. The candidate for the Linux system was a Compaq Pentium 166MHz machine with 64MB of RAM. It has a 1.2GB drive and a 4x ATAPI Sony CD-ROM. As a training system, multiple users needed access to this machine, so a LAN connection was required.

The network setup at my office is token ring. Network cables and a spare IBM token ring PCI card provided the connections. The system was placed in the computer center located on the first floor of the building. It resides behind some AIX mini-computers that cost over a quarter of a million dollars. The entire cost to the company for the Linux system's hardware was nil, since it came from another project. If the equipment had not been available, the total estimated hardware costs would have been, at most, $1500.

After the hardware installation, the system was ready for software installation. The Slackware version of Linux 2.0, from a CD-ROM, became the operating system of choice, because it provides many of the UNIX features the staff already uses in its core systems. The first step of software installation involved creating boot and root diskettes. Slackware provides several different installation options, depending on the hardware. The bare.i and color.gz files on the Slackware installation CD-ROM are the optimal choice for our setup. Once these files were copied to a hard drive on another computer, the **rawrite** command included on the CD-ROM was used to create the boot and root diskettes.

The boot diskette initiated the target system, which began loading a subset of the Linux operating system into memory. Next, the root diskette was loaded. This was enough to start the installation of the system.

The next step involved creating a native Linux partition on the hard drive, then loading the operating system. Slackware provides an easy way to do this with its setup process. This process is menu driven, and it allows you to install a mixture of utilities. The setup in this environment included the basic Linux system and X utilities.

After installing the software, configuring the system's startup was next. The startup routine was set to load the kernel from the hard drive. At that time, no network configuration took place, because Slackware requires you to recompile the kernel if you have a token-ring card.

At this point, the initial system was tested by running some of the non-network commands. After checking the system, the kernel was rebuilt for a token-ring network. Rebuilding the kernel so it would recognize an IBM token-ring card with a Tropic chip set was rather painless. This does require superuser access rights, however.

First, from the /usr/src/Linux directory, the **make config** command was run, starting a shell script that prompts the end user with questions to configure the operating system. The prompts usually default to the system's last kernel configuration. Below is the kernel modification for the token-ring card:

```
<<Token Ring driver support (CONFIG_TR)[N/y/?]-Y
 IBM Tropic chip set based adapter support
 (CONFIG_IBMTR)[N/y/m/?]-Y>>
```

No other commands need to be changed. Following the kernel configuration, the next four make commands must run for the system to recognize the changes:

```
make dep: make clean; make zImage; lilo
```

Briefly, these commands will create the necessary dependencies, remove object files, create the kernel image and allow the Linux loader to recognize the kernel. Creating the kernel image takes the most time. Depending on the machine, it could take as long as a couple of hours.

After the kernel was rebuilt, the file /etc/rc.d/rc.inet1 needed to be changed. This file loads all of the network addresses for the system. The Ethernet network setup was modified to a token ring by changing eth0 in the **ifconfig** command to tr0:

```
/sbin/ifconfig eth0 ${IPADDR} broadcast\
${BROADCAST} netmask ${NETMASK}
```

The rc.inet file was set with the appropriate IP addresses. The host file, /etc/hosts, was modified to provide an alias to some common systems. The entire system was tested using the **ping** command and by running a few TELNET sessions.

Although the system includes many of the X utilities, it was not set up to run the X Window System. These utilities are accessible from Windows NT workstations with X emulators, such as PCXware. Most users at the site run these utilities, since they are accessing this system using TELNET and browser sessions from their workstations running Windows NT 4.0.

The next step involved configuring an Apache web server for our Linux machine. We obtained the server from CD-ROM in a compressed format. The **uncompress** command was used to unpack the files in a directory called /usr/local/httpd. In our setup, end users needed to create and view their home pages. For example, I needed to be able to access my home page, called index.html, from my directory at /home/kitsukic/www/. This required a modification to the srm.conf file, which locates home pages and sets special parameters that affect servicing of end users. This file is located in the /var/lib/httpd/conf directory. In this scenario, change the value of **UserDir** from **local_dir** to **www**. Hence, to get to my home page, my call would be http://145.225.56.23:82/~kitsukic/. The server now allows a browser to access my home page in /home/kitsukic/www/.

In addition, the port number in the main server configuration file, /var/lib/httpd/conf/httpd.conf, needs to be changed from 80 to 82. The reason it needs to be changed is that another process uses port 80. Once the changes to the configuration files were done, the following command was activated in the /etc/rc.d/rc_httpd file:

```
/usr/sbin/httpd -f
```

This starts the Apache **httpd** server whenever the system is booted. Overall, the entire installation of the server was straightforward and did not require much effort. A mock user's web pages served to test the web server.

The final step of the installation involved creating processes that would make the system maintenance-free. The **cron** command provides the Linux user with this capability. The cron command runs backup and file-cleanup processes at specific times of the day. For the backup process, it runs a script that compresses and transfers essential files to another machine. Another process run by the cron command purges old log and trash files periodically. The two processes are somewhat maintenance-free. In order to create these scheduled jobs, the administrator must run the **crontab -e** routine from the root login, which provides a vi editor environment. Using this editor, the administrator can create a list of jobs for the cron command to run at specific times. For example, he or she could create an entry to tell users to log off the system every day at 6:00 PM in order to do backups at this time.

For training purposes, the system was loaded with C++ and Perl. Programmers can safely run C++ and Perl code without affecting the larger systems. The Linux system also hosts a group home page that links to the staff's web sites. There are also links to tutorials on how to create web pages. The main page also links to an experimental SQL database. It demonstrates to the user how to use HTML commands to connect and extract data from an SQL database.

Currently, the Linux system is open to anyone within the department who wishes to experiment with creating web-based products using C++, Perl and a UNIX operating system. Programmers have been creating web pages using HTML and Java. Several non-programmer analysts have used this system to start learning how to program in C++ and HTML. As of this date, there have been no system crashes. The system has been reliable and maintenance-free from the start.

As businesses look toward cutting costs, Information Systems managers in UNIX environments need to find creative solutions to train their staff. One alternative is using Linux. Although the stigma of being a hacker's non-supported operating system remains, Linux is surprisingly easy to install and maintain. It offers a rather inexpensive system, with many of the UNIX features common in bigger ones. This makes it attractive to managers trying to cut training costs while at the same time trying to keep their staff technically trained.

Charles Kitsuki is an Information Systems Development Manager for a telecommunication carrier. He leads a group of Programmer Analysts, Project Leaders, Business Analysts, Quality Assurance Trainer/Analysts and Supervisors

through the murky waters of maintaining and enhancing several software systems. When Charles is not busy trying to overcome the myriad of paperwork, he is writing programs and hacking on his operating systems. He can be reached via e-mail at kitsukic@pixi.com.

Archive Index Issue Table of Contents

Advanced search

New Products

Amy Kukuk

Issue #53, September 1998

Conix 3D Explorer, NetWinder, CommuniGate Pro Server 2.0 Beta and more.

Conix 3D Explorer

Conix Enterprises, Inc. has announced the release of Conix 3D Explorer for Linux ELF. While supporting standard Mathematica graphics, 3D Explorer also provides a new graphics type, GLGraphics, with extended graphics primitives and directives. New features include continuous surfaces, display lists, inline transformations and per-element control over graphics options. Standard Mathematica graphics can be converted to GLGraphics and enhanced, allowing you to build directly on your existing graphics skills. 3D Explorer comes with a run-time installation of OpenGL for Linux by Conix. 3D Explorer is currently available for Linux ELF, Windows 95/NT and PowerMac platforms.

Contact: Conix Enterprises, Inc., PO Box 4113, San Luis Obispo, CA 93403, Phone: 800-577-5505, E-mail: tech@conix3d.com, URL: http://conix3d.com/.

NetWinder

Corel Computer has announced the release of NetWinder, a network computer built upon the Linux operating system. There are three versions of NetWinder including The Netwinder WS (a scalable, out-of-the-box, web-server solution designed to enable small and medium-sized businesses to create a web presence), The Netwinder LC (especially designed for desktop use) and The

NetWinder DM (a development machine equipped with a 3GB hard drive and packed with Corel's development tools). More information about the NetWinder and Corel Computer is available at the company's web site.

Contact: Corel Computer, 150 Isabella Street, Suite 1000, Ottawa, OntarioK1S 1V7, Canada, Phone: 613-788-6000, Fax: 613-230-8300, E-mail: via web site, URL: http://www.corelcomputer.com/.

CommuniGate Pro Server 2.0 Beta

Stalker Software has announced the CommuniGate Pro Server 2.0 Beta, a platform-independent Internet messaging server. Key features include multi-platform, industry-strength administration via the Web, multi-domain support, anti-spam protection and unique IMAP multi-mailbox features. The CommuniGate Pro server can be configured, controlled and monitored from any computer connected to the Internet using any web browser application. On all platforms, the CommuniGate Pro presents the same interface and uses the same file formats, allowing any organization to switch server platforms in less than an hour. A CommuniGate Pro Server can be downloaded free of charge at http://www.stalker.com/CommuniGatePro/.

Contact: Stalker Software, 655 Redwood Highway, Suite 275, Mill Valley, CA 94941, Phone: 800-262-4722, E-mail: info@stalker.com, URL: http://www.stalker.com/.

J Street Mailer Release Two

InnoVal Systems Solutions has announced the release of a new production version of J Street Mailer Release Two, a full-function e-mail client written entirely in Java. J Street Mailer supports both POP3 and IMAP4 mail servers. One of the most useful new features is LDAP (Lightweight Directory Access Protocol). Other features include Preview Mail for examining mail on a POP3 or IMAP4 server, mulitple personas of signatures and prefaces within a single account for business and personal use, and virtual folders. The J Street Mailer is currently available for $49 US. Java Lobby members may obtain a license for $44 US. Students and faculty of accredited higher education and secondary schools may obtain J Street Mailer for $29 US.

Contact: InnoVal Systems Solutions, Inc., 600 Mamaroneck Avenue, Harrison, NY 10528, Phone: 914-835-3838, Fax: 914-835-3857, E-mail: innoval@ibm.net, URL: http://www.innoval.com/.

InterBase 4.0 for Red Hat Linux 4.2

InterBase Software Corporation has announced the release of InterBase 4.0 for Red Hat Linux 4.2. InterBase 4.0 for Linux is compatible with the commercial versions of InterBase on other platforms. InterBase includes InterClient, an all-Java JDBC driver, versioning architecture where readers and writers don't block each other, an active database that includes a full-featured trigger implementation, event alerters, tailored VAR program and UNICODE International Characters support. InterBase 4.0 for Red Hat 4.2 is freely downloadable from the InterBase web site.

Contact: InterBase Software Corporation, 100 Enterprise Way, Suite B2, Scotts Valley, CA 95066, Phone: 408-431-6500, Fax: 408-431-6510, E-mail: via on-line form, URL: http://interbase.com/.

Metro-X 4.3

Metro Link has released Metro-X 4.3, an X11 Release 6.3 server replacement with more speed, more features and a new low price of $39 US. Metro-X provides support for the fastest, most popular graphics cards on the market today. In addition, Metro-X includes touch screen support and multi-screen support at no extra charge. Metro-X uses a graphical configuration program for easy setup and a graphical adjustment tool for image placement on the monitor. Other features include hot-key exit, hot-key resolution switching, hardware panning and international keyboard support. Metro-X 4.3 can be ordered directly from Metro Link.

Contact: Metro Link Inc., 4711 Powerline Rd., Ft. Lauderdale, FL 33309, Phone: 954-938-0283, Fax: 954-938-1982, E-mail: sales@metrolink.com, URL: http://www.metrolink.com/.

Archive Index Issue Table of Contents

Advanced search

Driving One's Own Audio Device

Alessandro Rubini

Issue #53, September 1998

In this article Alessandro will show the design and implementation of a custom audio device, paying particular attention to the software driver. The driver, as usual, is developed as a kernel module. Even though Linux 2.2 will be out by the time you read this, the software described here works only with Linux 2.0 and the first few decades of 2.1 versions.

I'm a strange guy, and I want my computers to keep silent—that's why I wrote the “Visible-bell mini-howto”, where I suggest speakerectomy surgery be performed. On the other hand, I enjoy playing with the soldering iron to build irrelevant stuff. One of the most irrelevant things I ever conceived is recycling the computer's loudspeaker in a very-low-volume audio device. As you might imagine, the device plugs in the parallel port.

This article describes the driver for such a beast, shows interesting details of the kernel workings and is still short enough to be an easy text for almost any reader. A quick description of the hardware is mandatory, but you can safely skip over the first section and jump directly to the section called “Writing Data”.

The software described here, as well as the electrical drawing, is released according to the GPL and is available as sad-1.0.tar.gz (Standalone Audio Device) from ftp://ftp.systemy.it/pub/develop/, my own ftp site.

Part of this work has been sponsored by “SAD Trasporto Locale” (http://www.sad.it/), the bus company of Bolzano (Bozen), Italy. They plan to bring my hardware on their buses and renamed the company to match my package (smile). (See “Travelling Linux” by Maurizio Cachia, *LJ*, June 1997.)

Figure 1. Audio Device Schematic

The Underlying Hardware

My device plugs in the parallel port, and its schematics are depicted in Figure 1; The photograph under the tiele is the only model ever built (Italian buses will run a different flavour of such stuff, the “bus for bus”--ftp://ftp.systemy.it/pub/develop/b4b-X.YY.tar.gz).

I owe the basic idea to Michael Beck, author of the **pcsndrv** package; the idea sounds like “use the parallel data bits to output audio samples.” My own addition is “use the interrupt signal to strobe samples at the right pace.” Audio samples must flow at 8KHz and any not-so-ancient computer can sustain such an interrupt rate: my almost-ancient development box runs a 33 BogoMips processor and is perfectly happy playing parallel audio. The interrupt-based approach trades higher quality for increased hardware complexity than that needed by Michael's package.

As shown in the schematics, the device is made up of a simple D/A converter built with a few resistors; the signal is then reduced to 1.5V peak-to-peak amplitude and fed through a low-pass filter. The filter I chose is a switched-capacitor device driven by a square wave at ten times the cutoff frequency. The 6142 chip is a dual op-amp with rail-to-rail output, one of several possible choices for low-power single-supply equipment.

The output signal can be brought to a small loudspeaker, but can be listened to only in complete silence; other environments ask for some form of amplification. My preferred alternative to the amplifier is the oscilloscope, the typical hear-by-seeing approach.

Writing Data

The main role of an audio driver is pushing data through the audio device. Several kinds of audio devices exist, and the **sad** driver only implements the /dev/audio flavour: 8-bit samples flowing at a rate of 8KHz. Each data byte that gets written to /dev/audio should be fed to an 8-bit A/D converter; every 125 microseconds, a new data sample must replace the current one.

Timing issues should be managed by the driver, without intervention from the program writing out the audio data. The output buffer is the software tool that isolates timing issues from user programs.

In **sad**, the output buffer is allocated at load time using **get_free_pages**. This function allocates consecutive pages, a power of two of them; the *order* argument of the function specifies how many pages are requested and is used as a power of two. An *order* of 1, therefore, represents two pages and an *order* of 3 represents eight pages. The allocation order of the output buffer is stored

in the macro **OBUFFER_ORDER**, which is **0** in the distributed source file. This accounts for one page, which on the x86 processor corresponds to 4KB, or half a second worth of data.

The output buffer of sad is a circular buffer; the pointers **ohead** and **otail** represent its starting and ending points. The kernel uses unsigned long values to represent physical addresses, and the same convention is used in sad:

```
   static unsigned long obuffer = 0;
   static unsigned long volatile ohead, otail;
```

Note that the **ohead** and **otail** variables are declared as volatile to prevent the compiler from caching their value in processor registers. This is an important caution, as the variables will be modified at interrupt time, asynchronously with respect to the rest of the code.

We'll see later that sad has an input buffer as well; the overall buffer allocation consists of these lines, executed from within **init_module**:

```
   obuffer = __get_free_pages(GFP_KERNEL,
      OBUFFER_ORDER, 0 /* no dma */);
   ohead = otail = obuffer;
   ibuffer = __get_free_pages(GFP_KERNEL,
      IBUFFER_ORDER, 0 /* no dma */);
   ihead = itail = ibuffer;
   if (!ibuffer || !obuffer) { /* allocation failed
                                  */
   cleanup_module(); /* use your own function */
   return -ENOMEM;
   }
```

Any data that a process writes to the device is put in the circular buffer, as long as it fits. When the buffer is full, the writing process is put to sleep, waiting for some space to be freed.

Since the data samples flow out smoothly, the process will eventually be awakened to complete its **write** system call. Anyway, a good driver is prepared to deal with users hitting the **ctrl-C** and must deal with **SIGINT** and other signals.

The following lines are needed to put to sleep and awaken the current process, all the magic is hidden in **interruptible_sleep_on**:

```
   while (OBUFFER_FREE < OBUFFER_THRESHOLD) {
      interruptible_sleep_on(&outq);
      if (current->signal & ~current->blocked)
          /* tell the fs layer to handle it */
         /* a signal arrived */
         return -ERESTARTSYS;
   /* else, loop */
   }
   /* the following code writes to
    * the circular buffer */
```

What are **OBUFFER_FREE** and **OBUFFER_THRESHOLD**? They are two macros: the former accesses *ohead* and *otail* to find out how much free space is in the buffer; the latter is a simple constant, predefined to 1024, a pseudo-random number. The role of such a threshold is to preserve system resources by avoiding too frequent asleep**->**awake transitions.

If the threshold was 1, the process would need to be awakened as soon as one byte of the buffer was freed, but it would soon be put to sleep again. As a result, the process will always be running, consuming processing power and raising the machine load. A threshold of 1KB assures that when the process goes to sleep it will sleep for at least one tenth of a second, because it won't be awakened before 1KB of data flows through the audio device. You can recompile sad.c with a different threshold value to see how a small value keeps the processor busy. Too big a value can result in jumpy audio, i.e. the sound cuts in and out. The audio stream becomes jumpy because data continues to flow while the kernel schedules execution of the process writing audio data. The more heavily the computer is loaded, the more jumpy the audio is likely to be; if several processes are contending for the processor, the one playing audio might be awakened too late, after all pending data has been transferred to the audio device. In addition to lowering the wakeup threshold, you can also cure the problem by increasing the buffer size.

Naturally, the **write** device method is only half of the story; the other half is performed by the interrupt handler.

The Interrupt Handler

In sad, audio samples are strobed out by a hardware interrupt, which is reported to the processor every 125 microseconds. Each interrupt gets services by an ISR (interrupt service routine, also called “interrupt handler”), written in C. I won't go into the details of registering interrupt handlers here, as they have already been described in other “Kernel Korner” columns.

Managing several thousand interrupts per second is a non-negligible load for the processor (at least for slow processors like mine), so the driver only enables interrupt reporting when the device is opened and disables it on the last **close**.

What I'd like to show here is how data flows to the A/D converter. The code is quite easy, and the **OBUFFER_THRESHOLD** constant appears again, as expected:

```
if (!OBUFFER_EMPTY) { /* send a sample */
   OUTBYTE(*((u8 *)otail++));
   if (otail == obuffer + OBUFFER_SIZE)
      otail = obuffer; /* wrap */
   if (OBUFFER_FREE > OBUFFER_THRESHOLD)
      wake_up_interruptible(&outq);
```

```
        return;
    }
    wake_up_interruptible(&closeq);
```

As usual, every code snippet introduces new questions; this time you might wonder about **OUTBYTE** and **closeq**. The latter item is the main topic of the next section, while **OUTBYTE** hides the line of code that pushes a data sample to the D/A converter.

The macro is defined earlier in sad.c as follows:

```
#define OUTBYTE(b) outb(convert(b), sad_base)
```

Here, **sad_base** is the processor port used to send data to the parallel interface (usually 0x378), and **convert** is a simple mathematical conversion that turns the data byte as stored in the audio-file format to a linear 0-255 value, more suited to the D/A converter.

Blocking Close

The **close** system call, like **read** and **write**, is one of those calls that can block. For example, when you are done with the floppy drive, **close** blocks waiting for any data to be flushed to the physical device. This behaviour can be verified by running:

```
strace cp /boot/vmlinux /dev/fd0
```

Audio devices are somewhat similar to the floppy drive: a program writing audio data closes the file after the last **write** system call. However, this means only that data has been transferred to the output buffer, *not* that everything has necessarily already flown to the loudspeaker. An implementation that blocks on close can be helpful, when you want to do this:

```
cat file.au > /dev/sad && echo done
```

On the other hand, sometimes you'll prefer to stop playing sounds when the process closes the device. For example, if you play the piano on your keyboard, the sound should stop as soon as you raise the key, even if the program has already pushed extra data to the output buffer.

For this reason, the sad module implements two device entry points, one that blocks on close and one that doesn't block. Minor number 0 is the blocking device and minor number 1 is the non-blocking one. The entry points in /dev are created by the script that loads the module, included in the sad distribution: /dev/sad is the one that blocks on close and /dev/sadnb is the non-blocking one.

While real device drivers often offer configuration options (such as choosing whether or not to block on close) through the **ioctl** system call, I chose to offer different entry points in /dev, because this way I can use normal shell redirection to perform my tasks, without the need to write C code to perform the relevant ioctl call. The close method in sad.c, therefore, looks like the following:

```
if (MINOR(inode->i_rdev)==0) /* wait */
   interruptible_sleep_on(&closeq);
else {
   unsigned long flags; /* drop data */
   save_flags(flags);
   cli(); ohead=otail;
   restore_flags(flags);
}
MOD_DEC_USE_COUNT;
if (!MOD_IN_USE)
   SAD_IRQOFF(); /* disable irq */
return;
```

Actually, there is a third possibility as far as **close** is concerned: go on playing in the background as long as some data is there, even after the program has closed the audio device. This approach is left as an exercise to the reader, because I prefer having a chance to actively stop any device making noise.

Reading Data

Usually, a device can be read from as well as written to. Reading /dev/audio usually returns digitized data from a microphone, but I haven't been asked to provide this feature, and I have no real interest in hearing my voice.

When I built my first alpha release of the physical device, I found the need to time the interrupt rate, in order to be sure it was close enough to the expected 8KHz. (In the alpha version, I used a variable resistor to fine-tune the frequency, and I needed a way to check how it went.) The easiest solution that came to mind was to use the clock of the host computer to measure the time lapses.

To this end, I modified the interrupt handler so that it would write timestamps to an input buffer whenever the device was being read. The input buffer is a circular buffer just like the output buffer described above.

The previous excerpt from sad_interrupt showed that after writing an audio sample, the function returns to the caller. Any additional lines, therefore, are only executed if no audio data is there, so the rest of the ISR has thus been devoted to collecting timing information. This shows how I implemented “if there is no pending output, deal with input” rather than the more correct “if something is reading, give it some data.” This is acceptable as long as the device is not meant to be read from and written to at the same time in a production environment.

```
static struct timeval tv, tv_last;
unsigned long diff;
do_gettimeofday(&tv);
diff = (tv.tv_sec - tv_last.tv_sec) * 1000000 +
   (tv.tv_usec - tv_last.tv_usec);
tv_last = tv;
/* Write 16 bytes, assume bufsize
 * is a multiple of 16 */
ihead += sprintf((char *)ihead,"%15u\n",
            (int)diff);
if (ihead == ibuffer + IBUFFER_SIZE)
   ihead = ibuffer; /* wrap */
wake_up_interruptible(&inq); /*
        anyone reading? */
```

Printing the time difference between two samples has two advantages over printing the absolute time: data is directly meaningful to humans without resorting to external filters, and any overflow of the input buffer will have no effect on the perceived results, other than the loss of a few samples.

Real tests show the reported interrupt rate is not as steady as one would hope. Some system activities require you to disable interrupt reporting, and this introduces some delay in the execution of the ISR routine. Nonetheless, an oscillation of a few microseconds is perfectly acceptable and it is not perceived in the resulting audio, which is not high-fidelity anyway.

It's interesting to note that disk activity can introduce some real distortion in the audio stream, since servicing an IDE interrupt can take as long as two milliseconds (on my system). The IDE driver disables interrupt reporting while its own ISR is active, and the huge delay results in eight lost interrupts from the parallel port, which in turn causes a noticeable distortion of the audio data stream.

If you read from sad during disk activity, you'll see the long time intervals; writing to the device produces very bad audio. The easy solution to this problem is invoking

```
/sbin/hdparm -u 1 /dev/hda
```

before playing any audio. The command tells the disk drive not to disable reporting interrupts while it is servicing its own. Refer to the **hdparm** documentation to probe further.

Other Device Methods

The device driver interface offers other device methods in addition to the **open/close** and **read/write** pairs. While none of them is critical to device operation, I usually add a few lines of code to implement **select** and **lseek**. The former is needed by those programs which multiplex several input/output channels or use non-blocking operations to read and write data. Its role is quite needed if you run real programs, and the implementation is straightforward enough that

I won't show it here. The implementation of lseek, on the other hand, consists of the one line **return -ESPIPE;** and is meant to tell any program that tries to lseek the device that this “is a pipe” (reported to user space as “Illegal seek”).

Related Stuff

My aversion to computer sound makes me a novice in the field, and I really don't know anything about programs that play audio, or sites where audio files can be retrieved. Although Linus Torvalds offered an interesting “I pronounce Linux as Linux”, the file was not enough to test my device, and I needed to generate some audio data. The result is the sad distribution includes a program that plays sinusoidal waves, one that plays square waves and a not-so-good piano implementation. These tools work with any /dev/audio you happen to run and can be fun to play with, especially if you have a scope near your Linux box.

All code for the sad program is available by anonymous download in the file ftp.linuxjournal.com/pub/lj/listings/issue53/2997.tgz.

Alessandro Rubini tries to develop Open Source software/hardware for a living and that's why he and other hackers founded “Prosa Srl”. He can be reached at rubini@prosa.it, in addition to the usual addresses rubini@linux.it and rubini@systemy.it.

Archive Index Issue Table of Contents

Advanced search

MUP: Music Publisher

Bob van Poel

Issue #53, September 1998

Here's a look at notation editors for producing sheet music under Linux.

If you are a musician, you can only cry about the lack of music programs which run under Linux. Yes, there are many CD players and sound editors. However, when it comes to notation programs for producing printed sheet music, your choices are severely limited. My search for notation editors has turned up three choices: Rosegarden, TeX music systems and MUP (music publisher).

Rosegarden

The graphical program Rosegarden (http://www.bath.ac.uk/~masjpf/rose.html) is a very interesting program which tries its best to do everything. It has a notation editor which handles most of the normal editing functions, a MIDI sequencer which will play music from the notation editor as well as record data from a MIDI keyboard, and the ability to import MIDI files and convert them to notation—sounds wonderful. Unfortunately, Rosegarden is a work in progress and simply doesn't do all it is supposed to do, or does them awkwardly.

I have been unable to get the sequencer to work using my Gravis Ultra sound card, and I find that the notation editor is tedious to use, since there are no keyboard accelerators for entering note data. In addition, there is no easy way to print music. Rosegarden does have the option of exporting files in MusicTeX, OpusTeX and PMX (a preprocessor for MusiXTeX). I tried some of the combinations, but was not impressed by the output.

The biggest problem with Rosegarden (and a lot of other music editors) is that it works on the music as if it were a long string, which means changes to the start of the music propagate to the end of the chart. For example, if in bar one of a piece you have four quarter notes and you wish to change the first quarter note to two eighth notes, you change the first quarter to an eighth, then insert an eighth. When the first change is done, everything to the right of the edit

point is reformatted with the result that none of the music is now in the correct measure. Of course, inserting the second eighth fixes this. If you have several staves of music and you do a few edits, messing up the entire piece is much too easy.

TeX

I did not spend much time with any of the various TeX music systems. I can handle LaTeX for word processing, but the music variants seemed much too complex to use. All are in a beta state, and none produced output which looked finished to me.

MUP

MUP, at first glance, would probably be the last program to pick. However, after a fair bit of testing, I have decided to use it. So far, I'm happy with my choice. Quoting from the user's manual:

> The music publisher program called MUP takes a text file describing music as input, and generates PostScript output for printing that music. The input file can be created using your favorite text editor, or generated from any other source, such as another program. The input must be written in a special language designed especially for describing music.

Unlike Rosegarden (and the MS Windows offerings), MUP does not operate in a WYSIWYG environment. As a matter of fact, the MUP distribution doesn't even have a means of editing music. MUP uses plain text files with the appearance of source code as its input. Use vi, Emacs or whatever your flavor of editor is. Process the file with MUP to create postscript, and finally, print the postscript file. If you don't have a postscript printer, you'll need **ghostscript** to print things out, and **ghostview** is handy for screen previews.

As an example of how MUP uses lines of text to describe a piece of music, here are a few bars of music:

```
* 1: 8g;c+;e+;g+;g;b&c#+;g+;
* bar
* 1: 8g;b;d+;f+;4g+;g+;
* bar
* 1: 8g;c+;e+;g+;g;b&c#+;e+;
* bar
* 1: 4g+;b;c+;c#+;
* bar
* 1: 4d+;c+;a;f;
* bar
```

The **1:** at the start of each line is the staff/voice indicator (in this example, it refers to staff 1 and, since there is no additional argument, voice 1). Following the staff/voice are the notes for the measure. The first measure has an eighth

note g, eighth note c, etc. The next measure has several eighth notes as well as two quarter notes. At first this might seem to be a bit difficult to follow, but with practice it quickly makes sense. (See Figure 1 for example output.)

Figure 1. Printed Music Image

A MUP score can contain up to 32 staves of music, each with two voices. Each voice can have multiple notes (or chords), so complex arrangements are quite possible. In addition to the actual staves, you can also include lyrics, musical symbols and other appropriate items.

I started to use MUP when I was playing saxophone in a small combo. We all play from fake-type music (chords, lyrics and the melody line). Since I'm not the greatest sax player in the world and find it fairly hard to transpose from C to B flat while sight reading, I started rewriting the C charts into B flat by hand. I find anything needing a pen to be tedious, so I was inspired to try MUP. After doing a few practice charts, I am now able to enter a page of one-line music with lyrics in about an hour. Since MUP can produce MIDI files as well, I can create one in the right key for practicing at home.

Flushed with the success of doing these simple charts, I decided to try a more complex task. I also play in a 15-piece dance band. Most of the music we play is arranged by our leader, but recently some of the members have been doing some arranging as well. So, I decided to give it a try. My first arrangement of the old standard “Fever” took the better part of two days to complete—arranging it for 11 voices on 6 staves. We played it the other night and I was pleased—not only was everyone impressed by the appearance of the charts, it didn't sound bad either. The first page of the conductor's score is shown in Figure 2. The complete MUP files for “Bye Bye Blackbird” and “Fever” are available by anonymous download in the file ftp://ftp.linuxjournal.com/pub/lj/listings/issue53/3056.tgz.

Figure 2. Conductor's Score

If you would like to see some of my other arrangements, I have posted them along with a copy of this article at http://www.kootenay.com/~bvdpoel/.

I certainly don't have room in this short article to cover all the features of a complex program like MUP. A few of the more useful items I've been using are if/else statements to produce charts for different instruments, file includes to read in my own “boiler plate”, and macros to make my input files easier to create, read and revise.

MUP comes complete with a well-written, 99-page user's manual in PostScript (you'll have to print it out), as well as the same information in HTML format. Equally impressive is the customer support available via e-mail. I've sent a number of queries to the authors and have received courteous, timely replies to each and every one.

MUP is not free. You can download a working copy of the program, the source code and the manual from http://www.Arkkra.com/. In addition to the pre-compiled package for Linux, binary packages exist for other x86 UNIX systems capable of running ELF x86 binaries and a MS-DOS package. Also, the complete, commented source code is also available. This source should, according to the authors, compile on any platform with a C compiler. The Arkkra home site also has a pointer to a Macintosh port—this cross-platform support is a nice bonus as part of this excellent package. The program is a complete working copy—however, it prints a “this is an unregistered copy” watermark on all pages of the score. MUP registration is only $29 US; paying this gives you a license which turns off the marks. This is a fairly low price to pay for such a well thought out program.

This article was first published in Issue 28 of LinuxGazette.com, an on-line e-zine formerly published by Linux Journal.

Bob van der Poel (bvdpoel@kootenay.com) started using computers in 1982 when he purchased a Radio Shack Color Computer complete with 32KB of memory and a cassette tape recorder for storing programs and data. He has written and marketed many programs for the OS9 operating system. He lives with his wife, two cats and Tora (the wonder dog) on a small acreage in British Columbia, Canada spending his time gardening, practicing sax or just having fun.

Archive Index Issue Table of Contents

Advanced search

Best of Technical Support

Various

Issue #53, September 1998

Our experts answer your technical questions.

HP and Matrox

I have an HP Pavilion that comes with an on-board ATI video card. I now have a Matrox Mystic in the computer, and when I try to run X, I get the message “configured devices not found”. I think the HP is not recognizing the Matrox, but with the way the HP is set up the on-board disables itself when you install another video card. How can I make Red Hat 5 work correctly? Thanks a bunch.

—Fred Fredrickson, Red Hat 5.0

Sounds like the X server is still configured for the on-board video controller, which you say is now disabled. X won't find your new video card automatically. Try reconfiguring your X server for the new card.

—Scott Maxwell, s-max@pacbell.net

Login Names

Would it be possible to give a user a login name which consists of more than eight characters? For example, “fujigaki” and “mayuzumi” are quite popular family names in Japan. We wish to give them “hfujigaki” and “cmayuzumi” where the “h” and “c” are the first characters of their first names. In some UNIX systems, it is possible to do that. On the other hand, the default **adduser** command does not seem to support this extension. It would be great if this could be done.

—Tokuzo Shimada, Slackware 3.10

The latest glibc libraries (found in Red Hat Linux) allow a user name of up to 32 characters. The **adduser** (as well as other shadow utilities) script you've

mentioned does indeed have a limitation, but it should not be that hard to modify it or create your own.

—Mario Bello Bittencourt, mneto@buriti.com.br

Space Problem with X

Whenever I try to start X, I get an error message stating there isn't enough space in the /tmp directory. I've removed everything from that directory, but I still get the message. Is there any way to increase the size of this directory or at least force X to run? Thanks.

—Arnold Kelly, Caldera 1.2

You don't need to make space in the /tmp directory only—you can also free up space elsewhere on the same file system. On my computer (which is also running low on space), **df** reports the following:

```
% df /tmp
Filesystem 1024-blocks Used Available Capacity
Mounted on /dev/hda2 495746 468014 2129 100% /
```

This says that my /tmp directory is on the /dev/hda2 file system. Try the same with other directories, making space where you can in other directories on the same file system. In particular, I'd suggest checking /var/log, where your system logs are kept.

—Scott Maxwell, s-max@pacbell.net

Recent Hardware

Is there an up-to-date list of supported, recent hardware? How can I find out if Linux will run on a new computer configuration?

Your answer will determine whether I give my 486 to my brother-in-law or keep it as a Linux box.

Thanks for your time.

—Al Rivera, Slackware 3.4

You should check the Hardware-HOWTO on all the LDP mirror sites. However, I'm pretty suspicious of recent hardware; hardware manufacturers are always creating new stuff, and they rarely offer a Linux driver for their products. Linux support usually arrives later than the hardware product, but unless you play video games, you never need the processing power they're trying to sell you.

I prefer to buy my hardware from Linux-aware computer shops: that's the only way to be sure I won't be throwing the whole box in the wastebasket.

—Alessandro Rubini, rubini@linux.it

Printing Unformatted Text

I have an HP Laserjet 2p. The installation of Linux went well but when using **lpr** to print out a file, I get approximately three lines of unformatted text. At other times, I get no output at all. The entry in the printcap file that refers to the Laserjet yields two names: lp and hdj:\. When hdj is entered as printer name, the system states that device is unknown. When the **-P** option is used, printer problems persist, i.e., they are unchanged. I am a Linux novice and I could use some help. Thanks for your input.

—Dewey

This happens because the printer uses the DOS convention for newlines: **\r\n** (return, newline), while UNIX text has only a **\n** at the end of each line.

You should either filter the text through **unix2dos** (or through **sed**) or avoid sending unformatted text to the printer. I use the **a2ps** (ASCII to PostScript) filter and then **ghostscript** to convert PostScript to a PCL print file.

Setting up non-PostScript printers as if they were PostScript is quite easy with modern distributions. Look for the “Magic Filters” package and install it.

—Alessandro Rubini, rubini@linux.it

Recognizing New Memory

I have upgraded my PC's memory from 16MB to 72MB. Although the BIOS and Win95 recognize the extra memory, Linux appears not to see it. Using utilities such as **free** and **top** show a total of only 16MB of available memory. My PC still performs as it did when I had only 16MB installed.

Is there something I must do for Linux to recognize the new memory?

—George Tankoski, Slackware 1.3.2

LILO's configuration file may be explicitly setting the available memory to 16MB. To see if this is true, check the file /etc/lilo.conf for a line that looks like this:

```
append="mem=16m"
```

As root, change this line to read:

```
append="mem=72m"
```

Then, run **/sbin/lilo** (also as root) to make LILO reread the edited configuration file and reboot.

If you don't see the “16m” line in /etc/lilo.conf, back up your system, then try rebooting and typing the “72m” line directly at the LILO prompt. If your system boots and appears to be stable, you can then permanently enshrine the “72m” line in /etc/lilo.conf.

—Scott Maxwell, s-max@pacbell.net

Modem Connection to NT

I have a modem connection at work on a MS Windows NT/4 server. When I try to connect with PPP, I never get the login prompt. Once the dial up is done (using **seyon**), the connection hangs up. I guess NT is using RAS rather than the regular PPP. Do you know any way to set up the connection with NT?

Thanks for your help.

—Jacques Milman, Red Hat 4.1

The NT server uses ms-auth-chap authentication. Check out your **pppd** configuration to be sure it is compiled with ms-auth-chap crypted authentication.

Here is an example of what can happen if your pppd does not include ms-auth-chap support—server asks for ms-auth-chap:

```
pppd[164]: rcvd [LCP ConfReq id=0x0 <asyncmap 0x0>
<auth chap 80> <magic 0x307f> <pcomp> <accomp>]
```

pppd rejects the request:

```
pppd[164]: sent [LCP ConfRej id=0x0 <auth chap 80>]
```

so NT closes the line.

—Pierre Ficheux, pierre@rd.lectra.fr

Boot Problems

Let me start from the beginning. One day I booted my computer and received a message saying “remove and insert new disk” or something similar. I played

around with the CMOS setting and checked the hardware to make sure no wires had fallen out. I ended up formatting my c: disk, and up until this time, I could still use Linux. I received the same message, but I could load DOS from a boot disk.

The next time I tried to load Linux at the LILO prompt, it said “loading……..” and hung. I then tried to use a Linux boot disk and got the message “cannot initiate console”.

I am finding this to be a big problem, as I cannot access the files on either my Windows 95 partition or my Linux partition. If you can give me any help with fixing my Linux problem, I will be very grateful.

—Jamie Gamble, Slackware 3.2

The process of booting Linux on the PC platform is a bit intricate, mainly because of the peculiarities of the platform.

LILO loads the kernel using a list of disk blocks it built beforehand (when you ran **/sbin/lilo**, the map installer). After loading the blocks, it jumps to the kernel image; but if you moved the kernel after running /sbin/lilo, the loader will jump to nonsense program code, thus hanging the system.

Boot floppies, on the other hand, come in different flavours. The message “cannot open an initial console” means the kernel was loaded just right; it mounted a root file system, but couldn't open /dev/tty1 or /dev/ttyS0. I've seen this happen when mounting the /home partition as the root file system (there was no /dev directory).

Restoring a Linux installation is not trivial, especially if you have no other Linux box around. Short of finding a local Linuxer, try the /usr/doc/lilo*/README or my article “Booting the Kernel” in the June 1997 *LJ*.

—Alessandro Rubini, rubini@linux.it

Memory Exhausted

When I'm recompiling my kernel, I get a virtual memory exhausted error. It seems to happen when the compile is around the floppy.o section. When I use a different boot kernel (bare), it gives me a fatal signal 13.

—Wes Horn, Slackware 3.2

There are two common causes of this kind of problem.

1. Hardware; perhaps a bad cache or RAM
2. Out of swap space

You did not specify how much RAM nor how much swap space you have. Compiling the kernel is a CPU/memory intensive task, so if you do not have enough physical memory, your system will start using the swap space. If this swap is used up, strange things, such as the one you mentioned, can happen.

—Mario Bello Bittencourt, mneto@buriti.com.br

Metro-X Problem

Red Hat 5.0 comes bundled with Metro-X, a high-powered graphics server, and I have had a lot of problems getting it to run. A friend of mine running Linux said there is a bug in Red Hat 5.0 that prevents the necessary symbolic link from being created. I tried to correct it using the following command:

```
rm /etc/X11/X ln -s ../../usr/X11R6/bin/Xmetro \
   /etc/X11/X
```

This didn't solve the problem. When I try to start Metro-X my screen goes blank, and a few seconds later, the prompt returns with no error messages.

Can you help me solve this problem? Your help is greatly appreciated.

—Tim Perry, Red Hat 5.0

If that is exactly what you typed, you missed something crucial: a semicolon (;) before the **ln** (this separates the **rm** command from the **ln** command). Worse, you also removed the Xmetro binary!

To fix it, use **rpm** or **glint** to reinstall the Metro-X package, then re-run the command with a semicolon before the **ln**.

—Scott Maxwell, s-max@pacbell.net

Archive Index Issue Table of Contents

Advanced search