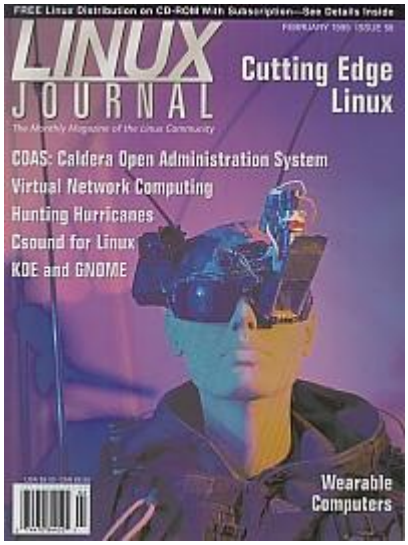
Linux Journal Issue #58/February 1999



Features

COAS: A Flexible Approach to System Administration Tools *by Olaf Kirch*

Caldera is working on a new easy-to-use configuration tool for Linux. Mr. Kirch gives us the details.

Csound for Linux *by David Phillips*

Mr. Phillips discusses some history as well as what's happening now in the Linux Csound world.

Hunting Hurricanes *by C. Wayne Wright and Edward J. Walsh*

The authors tell us about hunting hurricane using the Scanning Radar Altimeter based on the Linux system and analyzing the data with Yorick.

University of Toronto WearComp Linux Project *by Dr. Steve Mann*

Dr. Mann describes his WearComp (“Wearable Computer”) invention and how it has evolved into the same kind of philosophical basis for self determination and mastery over one's own destiny that is characteristic of the Linux operating system that currently runs on WearComp.

News & Articles

Virtual Network Computing *by Brian Harvey*

Mr. Harvey tells us about virtual network computing and how to set it up to control MS Windows Application from Linux.

Configuring ATM Networks *by Wayne J. Salamon*

COAS: A Flexible Approach to System Administration Tools

Olaf Kirch

Issue #58, February 1999

Caldera is working on a new easy-to-use configuration tool for Linux. Mr. Kirch gives us the details.

COAS stands for Caldera Open Administration System. It will be incorporated as the main configuration tool in future versions of the OpenLinux distribution.

For those who have never used OpenLinux, the tool we have been using for quite a while is called LISA (Linux Installation and System Administration), which is basically one huge shell script using a modified version of the **dialog** tool to interact with the user. When we felt it was time to move on to something new, we of course looked at what was already available. The only viable option at that time seemed to be LinuxConf, which had quite a ways to go before it would become useful. Since that time it has become much better, but because we had already started work on COAS, we decided to stick with it. Of course, we believe our concept is better.

The source code to COAS is released under the GNU General Public License. We feel our work might be useful to the Linux community as a whole and we want to invite interested programmers, administrators and users to participate in its development by offering comments contributing patches or even modules.

Vertical Modularity

The main idea behind COAS is not to provide just another administration tool, but an entire framework for writing one. From the start, we wanted it to be a modular application where assumptions about such things as system data representation, file locations and dependencies are separated as much as possible from user dialogs and vice versa. Ambitious as this goal may appear,

our main interest was the ability to easily adapt the tool to changes in the underlying platform and in porting it to other Linux platforms.

I like to call this vertical modularity, because it breaks up the task of system administration into three layers. At the lowest level are native system data files, such as /etc/passwd, /etc/hosts or files that define the IP address for a particular network interface.

On top of that, COAS implements an internal representation as a kind of database. If this term made you jump in your seat and shout, “Oh no, Mr. Bill, not a Linux Registry!”, please be assured that this is definitely not what we want it to be. COAS is supposed to be vi-administrator friendly. We want users to be able to switch between COAS and vi (or Emacs) administration, because even though we hope COAS will be useful for everyday tasks, it cannot cover each and every feature of a system component. (Consider the configuration monster incarnate, **sendmail**--you can spend as much time writing configuration software for it as Eric Allman keeps churning out new features.)

The native system files will remain the primary source of information. The COAS data model is strictly a run-time representation of system data that attempts to hide the on-disk representation from the upper layers. For instance, an administration module for the BIND server should not have to bother about where DNS zone files are located and how they are to be parsed; all it needs is the list of DNS zones this server is a primary or secondary name server for and the records they contain.

Having an abstract data representation also allows for alternate data access mechanisms. For example, our database engine can store a change log of an administration session to a file, which could then be distributed to other machines, thus allowing for bulk updates. Also, there's the vague idea that COAS might one day support remote access via LDAP or SNMP.

The top-most layer is the user interaction code. This code drives the dialog with the user and controls what information is displayed to the user at what time. It uses a standard set of dialogs, provides on-line help, etc. We decided to use a scripting language, Python, at this layer in order to allow for rapid prototyping. In addition to this, wrapping all lower-level functionality in Python classes and functions provides an additional level of insulation that restricts the number of tricks a programmer can pull. This may seem like a disadvantage to the hackers among you, but it is truly a big plus when it comes to code maintenance.

Horizontal Modularity

You may have guessed from my choice of the term “vertical modularity” that there is also a horizontal one, and so there is. Consider the following scenario:

a security problem or other misfeature requires you to update a component of your system, such as the BIND name server. Alas, the update is from version 4.9 to version 8.2, which uses an entirely different configuration file format. We could now ask you to install an all-new version of our administration tool in order to accommodate the new configuration file format. On one hand, that is costly in terms of bandwidth. On the other hand, making sure the tool operates properly with all possible combinations of updates applied or not applied would be rather time-consuming for us. The ideal solution would be to package the DNS server administration module alongside our BIND update.

We are attempting to accomplish the following: COAS lets you rip out an entire module, including the data model definition, Python code, message catalogs and so on, and replace it with a different version. We have nicknamed these CLAMs, which is short for Caldera Linux Administration Module (we invented the acronym first and then decided on its meaning, in case you were wondering).

Data Model

Let's take a closer look at the internal data representation. All information is stored in a tree, with each node having an individual name. For instance, the node containing the password of the root user is named system.accounts.users.0.password. If you're familiar with SNMP, think of the way SNMP variables are named.

Nodes can have different types; e.g., **system** is a directory, **users** is a list and **password** is a scalar. Scalar nodes can have various constraints attached to them; for instance, a string may be required to match a regular expression or contain only values from a predefined set of choices. You can also attach your own parsing and representation functions (written in C) to a scalar type, creating custom types that do such things as convert date strings, e.g., *Jun 12* or *tomorrow*, to UNIX time.

All this information is provided by the so-called schema. The schema acts as a sort of blueprint for the data model in much the same way an SNMP MIB definition describes the types and organization of entities for SNMP.

For instance, the definition of the mouse parameters might look like this:

```
MODULE   "PERIPHERALS/MOUSE"
MSGCATALOG "peripherals/mouse"
TYPEDEF DevicePathName  STRING MATCHES
    "/dev/[a-z0-9]*"
TYPEDEF MouseProtocol  STRING IN CHOICE {
    "Busmouse", "MouseSystems",
    "Mouseman", "Microsoft",
    ...
}
device         RECORD {
```

```
    model      STRING
    protocol    MouseProtocol
    deviceFile   DevicePathName DEFAULT
        "/dev/mouse"
    emulate3btn   BOOLEAN DEFAULT "false"
}
```

This creates a record named **mouse** containing five scalar nodes. For instance, **model** is a plain string variable, while **deviceFile** is a special string type whose definition is shown above the record. The first two lines contain some syntactic sugar that need not concern us at the moment.

```
%Those funky strings (|":MOUSE_EMULATE_NONE:"|)
%are tags for the COAS message catalogs.
```

This definition would be stored in a file named peripherals/mouse.schema (usually below /usr/lib/coas/schema) so that the mouse configuration would be accessible by the name **peripherals.mouse.device**.

When accessing data items, COAS instantiates the portions of the instance tree from the schema definition and populates the data by invoking so-called “mappers”. These mappers are responsible for parsing and writing back system files, locking them if necessary. Usually, they are written in C++ and kept in shared libraries loaded on demand. The most recent release also supports mappers written in Python.

In the case of the mouse device, there is no standard location where this information is stored. On a Red Hat box, for example, it is kept in /etc/sysconfig/mouse, a file which contains a list of shell variable assignments. COAS already has a general-purpose mapper for this type of file (it turns out that about 80% of all system files are quite close to four or five standard formats), so all that is left is defining the mapping. This is done by the so-called platform repository, where we might enter code like this:

```
peripherals.mouse.device {
    mapper     builtin.sysconfig
    path      /etc/sysconfig/mouse
    relation    MOUSETYPE:model:\
            PROTO:protocol:\
            DEVICE:deviceFile:\
            XEMU3:emulate3btn(map=/no=false,yes=true/)
}
```

The **mapper** keyword associates the mapper specified with the mouse device node. When accessing the device node, the first time, COAS detects this and invokes the mapper in order to populate the tree below the mouse device node. The mapper retrieves the **path** parameter and reads the file specified. The **relation** parameter tells the mapper which shell variables within the file correspond to which data model nodes.

The same thing happens when you have modified a protocol (e.g., the mouse) and invoke the device node's **commit** function. The data engine will invoke the mapper in order to write the data back to the file. Again, the mapper will use the specified **relation** to determine which data model values will be assigned to which shell variables. Note that in an act of vi-administrator friendliness, the mapper does not touch shell variables it does not know about and tries to preserve comments as well as it can.

The platform information is usually installed by merging it with the main COAS platform definition, which resides in /usr/lib/coas/repository.

User Interaction

Having written and installed the above files, you can already display and modify the mouse configuration using COAS. For example, COAS comes with small utilities such as **coas dump** and **coas change** that let you dump portions of the data tree or modify individual nodes. You can even write Python scripts that perform more complex operations on your data.

However, the ultimate goal (at least for us) is a Python module that interacts with the user, guiding him through the administration task. The module sits on top of the database engine and operates exclusively on the abstract data representation. It displays data to the user, selects which items are to be edited, provides on-line help, etc.

Why Python? Well, a very early prototype used Tcl as the scripting language, but for various reasons it didn't work too well. In contrast to Tcl, Python has fairly good object support and at least as good an extension mechanism. The other candidate was Perl, but we decided against it because it is so easy to write horrible code in Perl.

Communication with the user happens via an abstract user interface API written in C++, which currently supports a curses and a Qt front-end. Work on an extended Qt front-end that takes advantage of features provided by KDE is in progress. Of course, in order to be able to use this API from Python, a Python wrapper is provided.

The user interface provides a limited but useful set of dialogs: notice/question dialogs consisting of a text and a few buttons; list dialogs (single- and multi-selection, with or without check boxes, etc); prompt dialogs (containing edit fields for one or more scalar values); and table dialogs (which display data in a table, allowing in-place editing).

Listing 1

For instance, a minimal module for editing the mouse configuration would look like Listing 1 (some of the Python fluff, such as import statements, is not shown). For those not familiar with Python, this code defines the class **Mouse**, derived from the **CLAM** class defined in module **clam**. The **{__init__}** method is Python's way of declaring a constructor.

The method **run** is invoked by COAS. The first thing it does is look up the data model node for the mouse device. As described above, this step will trigger the parsing of the configuration file into the internal data representation.

Next, a prompt dialog is created and three edit fields are added for the mouse's model, protocol and device file. The last few lines are the somewhat standard dialog loop. Depending on whether the user terminates the dialog by pressing the Okay or Cancel button, either the **commit** or **cancel** method (inherited from the **CLAM** base class) is invoked, which displays a small question dialog along the lines of “Do you really want to save/cancel?”

What about Labels?

The first thing that will probably strike you as odd about this example is that it has no label strings anywhere. Nevertheless, the dialog is supposed to have a title, edit fields are supposed to have a label to their left, etc.

The answer is that COAS generates NLS strings for you out of the information it has. For instance, when creating the prompt dialog, we inconspicuously passed the string **mouse** into the function. As a consequence, COAS will create tags such as **|":MOUSE_TITLE:"|** for the dialog's title and attempt to look it up in the module's message catalog. (The message catalog name was specified in the class constructor.) Likewise, for the **protocol** edit field, it will generate the tag **|":MOUSE_PROTOCOL_LABEL:"|**. All you need to do is write the message catalog, mapping these funky strings to intelligible English (or French, German, etc.) and install the file.

Editing Data

Looking at the sample code above, you may also have thought: I understand where they put the data in the dialog, but how do they put it back into the data model?

This is the interesting part about the data editing process. If you have ever programmed Motif, you know how tedious it can be to extract the value to be edited from the data model, put it into the dialog and write the resulting value back to the data model when the user hits the OK button.

The approach taken by COAS is to tie data model nodes into the dialog directly and let the dialog select an appropriate widget type (string, combo box, toggle button, spin button, etc.). When the user provides a new value, the dialog will automatically check the value's syntax against data model constraints and write it back into the data model.

In our example, the dialog would create a simple string edit field for **deviceName**, a pop-up list for **protocol** (since it is limited to a set of choices) and a toggle button for **emulation**.

What's more, this mechanism offers you easy-to-use context help for each input field, bound to the **f2** key. Adding this type of help to a data item is as easy as adding the **HELP** attribute to the data definition in the schema file:

```
device          RECORD {
    model       STRING HELP "HELP_MODEL"
    protocol     MouseProtocol HELP "HELP_PROTOCOL"
    ...
}
```

These help messages will be looked up in the message catalog associated with the schema file (remember the **MSGCATALOG** keyword in the schema file?) and displayed in a pop-up dialog whenever the user presses **f2**.

Of course, every scheme you devise has a drawback. In this case, it is how to cancel changes made during the execution of the dialog. When the user presses the Cancel button, he wants all changes to go away.

This is where the **marker** object comes into play. The data node's **getMarker** method obtains a marker for the node's change log (called a journal in COAS lingo). When the user requests a discard of all changes, the CLAM base class invokes **self.mouse.cancel(marker)**, which reverts all changes made after the marker object was obtained.

Where's the Beer? er, Beef?

I have to admit that the above example, in its simplicity, is a bit deceptive. What I'm showing here is the simplest version of a dialog. In fact, what you see here is just a glorious interface to the configuration file because it does not offer the user any help or guidance. A good dialog would automatically choose the appropriate device file when a selection is made (e.g., a bus mouse) and keep the user from enabling three-button emulation for mice that already have three buttons. As a consequence, your average COAS module will have a lot more than those 20-odd lines in the example above.

However, the greatest advantage COAS offers in this context is that it relieves you of the usual hassle when working with a GUI and lets you concentrate on the data flow instead.

Dependency Model

Are You Curious?

If this article has piqued your interest and you would like to take a closer look at COAS, you can find out more about it on http://www.coas.org/ and http://developer.coas.org/. If you want to participate in the development of COAS, don't hesitate to contact me.

All listings referred to in this article are available by anonymous download in the file ftp.linuxjournal.com/pub/lj/listings/issue58/3019.tgz.

Olaf Kirch (okir@caldera.de) has been a Linux enthusiast since the MCC Interim days and has authored the *Linux Network Administrator's Guide* as well as various pieces of software for Linux. He has been the principal maintainer of the Linux NFS code for several years and has been working for Caldera since 1997.

Archive Index Issue Table of Contents

Advanced search

Csound for Linux

David Phillips

Issue #58, February 1999

Mr. Phillips discusses some history as well as what's happening now in the Linux Csound world.

Csound is a music composition and sound programming language originally written by Barry Vercoe at MIT. As Nick Bailey pointed out in his October 1998 *LJ* article “Sculptor: A Real Time Phase Vocoder”, Barry's original MUSIC11 program was eventually ported from PDP-11 assembler to UNIX C, where it became Csound. MUSIC11 was derived from the pioneering MusicV program by Max Mathews, perhaps the most revered “Founding Father” of computer music technology.

One of MusicV's major innovations was the implementation of the unit generator, a “black box” concept that allowed great extensibility to the language. A unit generator can be a signal generator or modifier, a patching opcode, a sensor, or it can provide sound file I/O and signal display types. Csound has evolved into a notable successor to Music V, quickly accommodating new synthesis methods and DSP algorithms. It is now at the cutting edge of modern computer music software. Linux Csound has done more than simply kept pace with that evolution—it offers capabilities not found with versions available on other platforms.

Enter Linux

In 1996, I wanted to try out the Linux OS. I knew certain software synthesis languages would run under it, and those languages were not available for DOS/Windows machines. Although Csound does indeed run under Microsoft operating systems (and many others), I was interested in seeing how well it would run under Linux. Jonathan Mohr had already added the real-time audio support for Linux, but I immediately discovered I had stumbled upon another big “DIY” (do it yourself) project. The source code available from the Bath, UK FTP site (the primary repository for the “canonical” packages) was a general

UNIX package, without Linux-specific Makefiles or any other compilation amenities. Although I was a novice at both Linux and the C programming language, I jumped in and started thrashing. With good assistance from John Fitch (maintainer of the Bath site and the canonical sources) and the helpful members of the Csound mail list, I finally produced a working set of Makefiles for the entire source tree. I soon had a fast Linux Csound with full support for X displays, real-time audio output and all the current opcodes. Professor Burton Beerman kindly provided an FTP site for my Linux Csound packages on his MusTec server at Bowling Green State University, and for two years I maintained the public version on that site and at Bath.

CSound in a Nutshell

Linux CSound: the Plot Thickens

Early in 1998, I received a message from Professor Nicola Bernardini at AIMI (Associazione di Informatica Musicale Italiana). He had thoroughly rewritten the Linux Csound Makefiles and wondered if I might be interested in adding them to the source package. His offer came at a good time, as I knew the code maintenance needed a more solid structure. Nicola's expertise was just the right factor appearing at just the right moment. His Makefiles enabled me to quickly prepare a variety of distribution packages (with or without X support, static build or shared lib, real-time audio enabled/disabled, etc.) and compile a more complete build of the source tree. Most importantly, the Makefiles created libcsound.so, a shared library which drastically reduced the binary's memory footprint (from about 450KB to less than 20KB).

Real-time Linux CSound

Around the same time, developer Gabriel Maldonado wrote a set of MIDI output opcodes, allowing Csound to be used as a MIDI composition/control instrument. Csound already accommodated MIDI input, directly from /dev/midi or from a Type 0 Standard MIDI File (see Real-time Midi Input). Gabriel's opcodes are different: they permit exploration into MIDI composition algorithms simultaneously with the rest of Csound's real-time I/O. Hypothetically, it would be possible to have a MIDI device controlling one Csound instrument while another instrument sends its output to **devaudio**. Given support for a full-duplex sound card, it should even be possible to have asynchronous I/O for both the MIDI and the audio ports.

Alas, no routines had been written for Linux Csound that would accept the data from Gabriel's opcodes and send it out to the MIDI port. After studying John Fitch's code for the Windows Csound MIDI output handler, I decided to try writing the appropriate calls for Linux. I fumbled around with the OSS/Free API and eventually wrote the code needed to activate the requested MIDI interface

and accept the control data sent to it from the Maldonado opcodes. Linux Csound was as up-to-date as any other version, and the necessary code for MIDI output had been trivial to write, consisting primarily of a few calls to the sound card API macros.

The CVS Repository

The next major step taken for Linux Csound was the establishment of a CVS repository. I had been complaining to Nicola that I found myself constantly checking everything coming to me in the canonical UNIX package, when he suggested the need for a revision control system. He volunteered to set it up at AIMI and after some trial-and-error hacking, he established the system we work with today. The CVS repository maintains separate directories for the canonical sources and the Linux-specific code. In this way, we can avoid rewriting sources just for Linux and we are always able to refer back to the “untouched” originals. Anonymous access to the CVS is permitted, but submissions for changes are carefully screened by the maintainer.

The Csound UNIX/Linux Development Group

Of course, a CVS development repository isn't of much use unless it has developers contributing, so a logical next step was the formation of the Csound UNIX Development group. Programmers Robin Whittle and Damien Miller joined in immediately, and Damien kindly provided a web page with all pertinent information for anyone interested in joining the group. It is worth noting that the group is for development, not just developers. We welcome anyone interested in seeing Linux Csound grow into the finest language of its kind. Programmers are certainly welcome, but so are musicians, audiophiles, DSP engineers and anyone else with an interest in Csound and its possibilities.

In October 1998, two new members made significant contributions to the group's activities. Gabriel Maldonado donated his entire source tree to the CVS repository, which enables Linux Csound to keep up with the developments for his Windows versions. (This generosity is quite typical of the Csound community. Much code sharing occurs on the Csound mail list, with new instrument designs freely offered, along with much healthy debate over various computer music issues.) The other signal addition has been Fred Floberg, whose contributions require special description.

Csound's internal support for real-time audio has been dependent on calls to the API for the OSS sound-card drivers. While certainly sufficient for casual use, many sonic notions such as full-duplex and multiplexed real-time audio I/O are not realizable by the OSS/Free driver. However, the ALSA driver does indeed support those uses; thanks to code from Fred Floberg, Linux Csound now explicitly supports the ALSA interface. (The ALSA project, led by Jaroslav Kycela,

is forming a new extended sound system API compatible with OSS/Free, but permitting much more advanced uses for sound-card features not supported by OSS/Free.) Fred is currently working on expanding MIDI file support. Csound now supports only Type 0 MIDI files, but Linux Csound should soon support the Type 1 and perhaps even the Type 2 Standard MIDI File formats.

Also, thanks to Robin and Damien, the Linux Csound distribution now supports the popular RPM packaging and can be built for glibc (libc6) systems. Debian users will be pleased to note that developer G<\#252>nter Geiger has prepared a package in the DEB format. Finally, Nicola Bernardini has written a Csound orchestra (instrument design) parser, which we hope will eventually be absorbed into the package. Such a utility is most helpful to a GUI designer, which brings me to my next topic: the power of Linux Csound and X.

The X Picture

My Linux soundapps web page shows more than twenty entries in the "Csound Helpers" section. The brief descriptions which follow are just that—brief descriptions which in no way indicate the full power of these applications. The examples shown here are for Linux systems running X; some excellent command-line utilities exist too and are included on the Linux soundapps page for those tools. All of these utilities work with the current versions of Linux Csound (3.47 or higher).

Note that each of these applications was built using freely available tools. The GNU C and C++ compilers, Tcl/Tk, Java, LessTif and WINE are powerful allies in the advancement of Linux sound and MIDI software. Their developers are to be commended for the wonderful work they have done for the good of the Linux community.

Cecilia

Cecilia (by Jean Piche and Alexandre Burton at the University of Montréal) is a fully-developed Csound composition and sound-processing environment. Written entirely in Tcl/Tk, Cecilia utilizes the entire range of possibilities afforded by Linux Csound, presenting a beautiful graphic interface (customizable, of course) and a powerful composition language (Cybil). Numerous real-time controls are supported, nearly all aspects of the program are user-definable, excellent on-line help is available and the GUI fully exploits Tk in the X environment. Cecilia won first place in the awards for computer-aided composition and realization software at the 1997 Second International Music Software Competition in Bourges. (See Figure 1.)

Figure 1. Cecilia

Rain

At the other end of the scale is developer Matti Koskinen's **rain**, a GIF-to-Csound score converter. A Csound score is the control file for a Csound instrument, providing it with such values as event start times, durations, amplitudes and frequencies, waveform selection and so forth. Matti's utility simply takes a GIF image, applies some user-defined values and magically translates it into a Csound score. The score can then be synthesized and played from within the application, or it can be saved to disk for later processing (perhaps in Cecilia). (See Figure 2.)

Figure 2. Rain

Adsyn

Adsyn is a graphic editor for Csound "hetro" analysis data files. **hetro** is one of the Csound sound file utility programs and its operation is quite simple. Using a heterodyne filter bank, it analyzes a sound file and creates a data file of separated frequency and amplitude values. That data file can be read and graphically represented by Adsyn and the frequency and amplitude components can be freely altered using the mouse. Csound's resynthesis opcode (adsyn) can be called; the edited file can then be synthesized and played from within Adsyn. Professor Oyvind Hammer originally wrote Adsyn for SGI machines at NoTAM, a Norwegian center for music and acoustics research. With his good graces, I began the port to Linux. It was finished with much assistance from Nicola Bernardini. (See Figure 3.)

Figure 3. Adsyn

Ceres2

Ceres2 is Johnathan Lee's enhanced version of Oyvind Hammer's Ceres, described in my September 1998 *LJ* article "Porting SGI Audio Applications to Linux". Johnathan greatly extended the editing capabilities of the original software engine, which essentially performs a Fast Fourier Transform (FFT) on a sound file and renders a graphic representation of its frequency content and activity. The graphic display can be edited in various ways, a large number of transforms (spectral mutations) are available, up to three graphic linear control functions may be specified and a variety of output formats are supported, including two types of Csound scores. Ceres2 also extends some of the command-line analysis variables such as FFT size, analysis window size and window overlap. The Linux port was done by me, but it was dependent on work already done on the original Ceres with great help from Richard Kent, who also supplied the invaluable tichstuff libraries which replace the SGI libs. (See Figure 4.)

Figure 4. Ceres2

Rosegarden

The Rosegarden suite includes a MIDI sequencer, a common-practice music notation display and the very nice feature of being able to save your work as a Csound score file. Such a tool is especially helpful for users most comfortable with standard notation conventions, allowing them to compose with their familiar symbols and then easily convert their creations for use with Csound instruments. (See Figure 5.)

Figure 5. Rosegarden

HPKComposer

The Java programming language lends itself to the easy creation of platform-neutral user interfaces. Jean-Pierre Lemoine's HPKComposer is an excellent example of a “pure Java” application, running under Windows, Mac OS and UNIX variants. Preparation for Linux is straightforward, depending upon successful installation of the Java development environment (JDK) in version 1.1.6 or higher, the Swing class libraries (version 1.1 beta3) from Sun Microsystems and Csound. HPKComposer blends aspects of the CMask program with the synthesis and DSP methods of Csound: tendency masks are used to create composition algorithms, which are realized by the synthesis engines (opcodes) of Csound. VRML displays are supported, the program is user-extensible, and although Java's current sound support is limited to 8-bit 8 kHz audio, when JDK 1.2 arrives it will support 16-bit 44.1 kHz CD-quality sound. (See Figure 6.)

Figure 6. HPKComposer

PatchWork

Russell Pinkston's PatchWork for Win95 is a graphic “patcher” for the design of Csound instruments. Although a UNIX/Linux version of this program exists (XPatchWork), it has not been maintained and is in need of some serious debugging. However, the Linux WINE Windows emulator can run the Win95 version, proving once again that Linux always finds a way. (See Figure 7.)

Figure 7. PatchWork

SoundSpace

Developer Richard Karpen has generously shared many of his opcodes with the general Csound community, one of which is called “space”. In the Csound manual entry for space is a mention of a GUI for creating the values needed by

the GEN28 stored-function table, and SoundSpace is that GUI. Written in core Java, this unique utility provides a visual interface for determining the placement and sonic trajectories of up to 8 sound files in the auditory space, with support for stereo and 4-channel output. (See Figure 8.)

Figure 8. SoundSpace

Into the Future

What is still to come? By the time this article is published, I hope to have some more Csound/Java applications running. Developer Michael Gogins has expressed great interest in seeing his “Silence” Csound environment running under Linux Java, and the prestigious IRCAM Music and Sound Research Center announced that a Linux version of their MAX for Java will be available at the end of 1998. Who knows; maybe someday I'll get around to completing my Tcl/Tk clone of Csounder, the popular Csound “launcher” for Windows (or at least get it working better under WINE).

The most recent versions of Linux Csound (3.49.xx and up) can be built for use on the 64-bit DEC Alpha. Thanks to developer Ed Hall, Linux Csound can claim to be the first 64-bit music and sound composition language widely and freely available to the public.

Nicola Bernardini continues to improve the distribution packaging: building Linux Csound is easier than ever, thanks to his incorporation of the **configure** utility. Work proceeds on accommodating **autoconf** and **automake**, since it is a primary objective to use the best tools available for creating the best possible distribution.

One of the intriguing problems facing the development group is how to make Csound re-entrant, enabling a plug-in architecture for Csound. To many of us, such an undertaking would mean a complete rewrite of Csound, and who knows where that might lead—“Son of Linux Csound”, perhaps? If you would like to join a very interesting distributed development project, take a look at the links listed in Resources and feel free to join the development group mail lists.

Richard Boulanger is a professor at the Music Synthesis Department of the Berklee College of Music. In the spring of 1999, his Csound book will at last be published by MIT Press. On one of the included CDs, you will find an article (which will, of course, be out of date by then) about running Csound under Linux. Yes, it was written by me, but I don't mention it to blow my own horn. This book is a massive tome and it includes contributions from all the major (and some not-so-major) members of the international Csound community. It should inspire many new users, several of whom will discover for the first time that Csound is available on the Linux platform.

Final Words

Linux Csound offers terrific possibilities for real-time computer music performance. Along with advances in real-time support, Linux Csound can be expected to stay at the cutting edge of synthesis methodologies, interface design, DSP algorithms and composition strategies. It is an ideal tool for contemporary sonic exploration and it demonstrates once again the flexibility and power of Linux, the cutting edge OS for the modern musician.

Resources

David Phillips (dlphilp@bright.net) is a composer/performer living in Ohio. 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. Dave also enjoys reading Latin poetry, practicing t'ai-chi-ch'uan, and any time spent with his lovely partner Ivy Maria.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Hunting Hurricanes

C. Wayne Wright

Edward J. Walsh

Issue #58, February 1999

The authors tell us about hunting hurricane using the Scanning Radar Altimeter based on the Linux system and analyzing the data with Yorick.

Figure 1. Front View of NOAA-43, One of Two WP3Ds

In March 1998, we started development of a new Linux-based data system for the NASA Goddard Space Flight Center scanning radar altimeter (SRA). The goal was to significantly reduce the system weight and volume to enable its installation on one of the NOAA hurricane hunter WP3D aircraft (see Figure 1) for the 1998 hurricane season. The SRA measures hurricane directional wave spectra and storm surge. The data will ultimately be used to help refine and improve hurricane models and improve forecasting and understanding.

The 1998 hurricane season was quite active and the SRA successfully flew in hurricanes Bonnie, Earl and Georges, collecting almost 50 hours of actual mission data.

Our principal obstacle was the short time frame until we needed to be operational onboard the hurricane hunter. The size, weight, complexity and power consumption of the SRA were also critical design items because of floor loading considerations and the limited payload capacity of the P3 aircraft when operating on long (10-hour) missions in turbulent weather conditions (hurricane-eye wall penetrations). Interrupt response time, crash-proofness and freedom from “lock-ups” were all important considerations when choosing the operating system for the SRA.

The new SRA data system, built on top of a Red Hat 4.2 system and Linux kernel 2.0.29, occupies eight inches of vertical rack space, weighs about 40 lbs, runs totally from an internal 12-volt aircraft battery and requires about 120 watts of

total input power. It includes a custom ISA board with several PIC microchips which perform dedicated functions for the radar. It also includes the entire radar IF (intermediate frequency) strip, detectors and a 2ns/point waveform digitizer. No monitor or keyboard is directly connected to the SRA; instead, Linux laptops are used for all control and display. Those laptops run Red Hat 5.1 and 2.0 Linux.

The RT-Linux (Real-time Linux) software does the following:

- Drives the waveform digitizer.
- Computes the centroid-based range measurement between the transmit and return pulses.
- Manages 96 automatic gain control loops.
- Corrects for aircraft attitude and off-nadir angle.
- Deposits formatted data in a shared memory block from which a normal Linux program extracts and records it to a disk file.

The SRA makes extensive use of Tcl/Tk and the Blt graphics library for real-time display.

Post-processing of SRA data is done with Yorick, a free and very powerful programming language that runs on Linux, a wide variety of other UNIX platforms and MS Windows.

Background

The previous implementation of the SRA was developed in 1988 using an array of 68020s on a Multi-bus-I backplane, a CAMAC crate full of nuclear physics instrumentation and a combination of UNIX and VRTX (VRTX is a real-time kernel). VRTX ran on real-time processors and UNIX ran on the system host. The CAMAC crate was quite heavy, consumed considerable power, occupied substantial rack space and was expensive. It used hardware time-interval units (TIUS) to measure the time for a radar pulse to travel from the aircraft to the ocean and back. It used “threshold detection”, which caused the TIU to stop and a CAMAC-based waveform digitizer to acquire the return waveform. The waveform data required its own 68020 processor to “process” each waveform and extract certain data. The data were used to refine (post-flight) the range measurement made by the TIU. Threshold TIUs suffer from an effect known as “range walk”, which causes the measured range to vary as a function of the strength of the return pulse. The array of processors communicated with each other via a 4MB memory card which resided on the multi-bus. Control of the system was via a character-based terminal and real-time display was done on an SBX Matrox graphics module which was managed by its own 68020 processor. One of the 68020 processors ran UNIX; that processor ran programs

which extracted radar data from the 4MB card and stored it on a 9-track magnetic tape or a disk file. The UNIX processor hosted all software development and managed the operator control terminal.

Due to its volume, weight and power consumption, we were unable to install this version of the SRA on the hurricane hunter. Limitations in the hardware signal-tracking circuits would frequently falsely trigger the system on a side lobe and effectively eliminate the true range measurement.

SRA System Description

Figure 2. Block Diagram of the SRA Sensor

The SRA is an airborne, 36GHz, down-looking, raster-scanning pulsed radar. A simple schematic block diagram of the sensor is shown in Figure 2. Its one-degree beam (two-way) is scanned across the aircraft flight track and a precise time-of-flight measurement is made for each of 64 pulses transmitted at 0.7 degree intervals across the scan. As the aircraft proceeds, a topographic image of the surface (normally ocean waves) is developed, recorded and displayed. The nominal ranging accuracy of the SRA is 10cm. Three differential carrier-phase GPS receivers are used to measure the exact location of three GPS antennas mounted in an array on top of the aircraft. A ground-reference GPS is set up where the flight originates and the ground and aircraft GPS data are processed post-flight to produce an aircraft trajectory, typically accurate to about 30cm in our application. Higher accuracies are possible when operating under less stressful flight conditions.

Figure 3. The SRA Scanner Assembly Mounted on the NOAA P3

The Radar Components

The SRA radar consists of a 20-inch Rexalite lens, a feed horn on the lens axis which looks up into a mechanical scanning mirror that mirror-images the feed horn to the focal point of the lens, a pulse modulator and RF exciter, receiver, 1.7KW Extended Interaction Amplifier (EIK) and the RT-Linux data system. The data system is the topic we will discuss here. Figure 3 is a photograph of the SRA scanner installed on the NOAA hurricane hunter. The fairing is removed in this photo.

Figure 4. SRA Data Power System

Figure 4 is a block diagram of the SRA power system. The SRA requires an uninterruptible power source for Linux and the three differential GPS receivers and computers. Instead of an off-the-shelf UPS, we went with a 12-volt 25 AH

“RG” (recombinant-gas) sealed aircraft battery as the prime power source for the system. This was chosen for two reasons:

- We needed an uninterruptible power source, because aircraft are notorious for power dropouts during engine start and shutdown.
- We needed to power our 12-volt GPS receivers for up to an hour before and after each mission without aircraft power applied.

We purchased a 12-volt input 150W PC power supply to power the data system. The battery can power the data system and the three GPS receivers for about two hours, or the GPS receivers alone for five hours. We located the battery in the rear of the custom data system housing.

Figure 4 depicts the wiring of our power system.

Figure 5. Block Diagram of the SRA Data System

Figure 5 is a block diagram depicting the internals of the SRA data system. The computer is a single-board 200 MHz Pentium which plugs into a passive backplane with ISA and PCI slots. The CPU card contains PCI-VGA video, PCI-IDE controller, PCI fast-wide SCSI controller, 64MB of RAM, 512MB of cache, two serial ports, a parallel port and the CPU. A PCI 3c595 network card provides networking and a special-purpose ISA card loaded with PIC microcontrollers provides an interface to the radar systems. A 6.4GB EIDE disk drive is used as /dev/hda to hold Linux and for data storage. A backup SparQ 1.0GB removable drive is installed as /dev/hdb. The system has no floppy or CD-ROM drive. If a CD or floppy is needed, they are simply remotely mounted with NFS from one of the Linux laptops which have both. No keyboard or monitor is used for normal operations, though they can be plugged in if the need arises.

Figure 6. The SRA Data System during Development

Figure 7. View of SRA Data System Internals

Initially, we used a 4.2GB SCSI drive, but that used too much electrical power. Early development was done using a 250W 117vac PC power supply. When we switched to the 12-volt input 150W power supply, we discovered we were over our power budget by 25 watts or so. During the boot process, the power consumed by the combination of the SCSI drive and the waveform digitizer would cause the power supply to “spike” the 5-volt source and cause a reboot. It took us several hours to find this problem. It would generally happen just as Linux began loading, due to the digitizer being powered up and the drive being accessed. During the DOS boot, the digitizer was not powered until after DOS booted and after the digitizer configuration program loaded and ran. Consequently, the loading of Linux was the straw that broke the camel's back.

We finally settled on a 6.4GB EIDE disk drive for Linux and for data storage. The power consumption of the EIDE drive is substantially less than the SCSI and no perceptible difference is seen in performance of the data system.

Figure 6 is a photo of the SRA data system during development. It was in this "state" until just a few days before our first test flight on the NASA C-130. Figure 8 is a photo of the data system after packaging. Figure 7 shows the internal organization of the data system as viewed from the top rear. The enclosure is reversed from most rack mounts. We wanted to have ready access to the computer card connections without having to remove the rear rack cover. The only connections on the rear are for the GPS receivers and the battery charger. The black power supply under the data system in Figure 8 is our prime power supply/battery charger.

Figure 8. Data System after Packaging

2ns PCI Waveform Digitizer, DOS, GageScope

The core data acquisition device in the SRA is the GageScope 8500-PCI waveform digitizer. It provides for up to 128KB of sequential samples taken every 2ns (nanoseconds). This permits us to digitize a 256-microsecond waveform. We actually digitize for 60 microseconds, beginning a few hundred nanoseconds before the radar pulse is transmitted and ending after enough time has expired to accommodate a signal return from our highest possible altitude. The pulse takes 2 microseconds to travel 1000 feet and return, so to accommodate a maximum altitude of 30,000 feet, we need to digitize at least 60 microseconds. Since a point is digitized every 2ns, there will be 30,000 points in each waveform. We don't read all 30,000 points out of the digitizer. We "track" the position of the returns and read out only 256 points centered around where we expect the return to come from. Since the ocean is basically flat, this technique works well.

The driver code provided by Gage for the 8500 supports DOS, Windows and Windows NT. It is extensive, to say the least. It contains several thousand lines of code solely to initialize most of the cards that Gage makes to an operational state. Apparently, much of the functionality of the card is loaded into programmable Logic Arrays from the DOS driver. The Gage driver code supports virtually every waveform digitizer made on several different OS platforms. They make extensive use of conditional compilation to select both the desired digitizer board and the desired operating system. They attempt to establish an isolating layer of driver code, so that a common set of driver calls appears to the users of their supplied library.

After looking at the driver start-up code, we though it might take more time to port the start-up code to Linux than we could afford. In order to avoid porting

the long and complex start-up code, we elected to make the system dual boot Linux and DOS. This scenario has worked well, permitting us to get a DOS program going quickly which would configure the digitizer. After the digitizer is configured, the autoexec.bat DOS script loads Linux using **loadlin**, a DOS program which can load a Linux kernel from DOS. The DOS digitizer start-up code leaves the digitizer in a known functional state. The code required to use the digitizer is actually not very extensive and only requires accessing a few registers and memory locations on the Gage card. The folks at Gage were very helpful in getting it working.

Hardware Interrupts

Waveform data are extracted from the digitizer after a radar pulse event has occurred. One of the 16C65A microcontrollers controls all aspects of triggering the transmitter, actuating various gates, triggering the waveform digitizer and finally interrupting the Linux waveform digitizer interrupt handler.

Figure 9. RT-Linux Interrupt Jitter

The RT-Linux interrupt typically responds in 2 microseconds (on our 200MHz Pentium) with occasional jitter to several microseconds. When we did this same test with MS Windows a couple of years ago, we found the fastest response to be on the order of 50 microseconds (486-dx2 66MHz) with jitter well into tens of milliseconds. It is incredible just how responsive Linux is to interrupts. Figure 9 is a digital scope capture, where the top trace rising edge is the actual hardware interrupt signal on the ISA backplane. The bottom trace is a hardware signal generated by the interrupt code. It simply wrote a “1”, waited awhile, then wrote a “0” to the printer port. Each horizontal division is 2 microseconds. This demonstrates the typical latency of our RT-Linux system. Concurrent with this test, we ran a **find** command in another xterm so the system had something to do.

Microcontroller Board, 16c65a

Microcontrollers permit very hard and reliable real-time capabilities. They are well-suited to replacing arrays of chips and digital logic in many applications such as the SRA. We designed a special ISA interface board for the SRA which encompasses most of the special requirements of the radar and special interfaces.

The board is presently populated with four Microchip 16C65A microcontrollers. One microcontroller implements a real-time clock which automatically maintains time-of-day synchronization with our GPS receivers. It has a least significant fractional time bit of 200 nanoseconds and provides the SRA with up to 64 bits of accurate time information. This chip automatically captures the

trigger time for each radar pulse. It and its neighbors can all be read and written by Linux.

A pair of microchips function together to convert the scan encoder pulse trains into radar trigger events. As our scan mirror rotates, a scan encoder measures the scan angle. At our scan rates, it produces a pair of 40KHz square-wave signals which are 90 degrees out of phase. One microchip is programmed to combine these two signals together and produce a single 80KHz signal, which is then counted to determine the position of the scanner. The second microchip is programmed to count the 80KHz signal and initiate a radar pulse at predefined angles. With its 200ns instruction time, this microchip directly controls all aspects of the transmitter and receiver electronics and also generates an interrupt to Linux once a waveform has been acquired by the waveform digitizer. For each SRA pulse, this microchip:

- Protects the receiver front-end from damage.
- Verifies that the receiver is protected.
- Gates the digitizer on.
- Gates the EIK amplifier on.
- Delays exactly 200ns.
- Triggers the transmitter modulator to generate an 8ns pulse and causes the real-time clock microchip to capture the present time.
- Delays 200ns for the transmit pulse to be well clear.
- Enables the receiver to receive return signals.
- Interrupts the RT-Linux SRA module to extract the waveform data from the digitizer.

RT-Linux

RT-Linux is a patch which gives Linux many of the most important features needed by real-time programmers and embedded-system designers. It is implemented as a set of modules which can be installed and removed using **insmod** and company. You also use insmod to install any real-time code you write. RT-Linux programs execute in the kernel space and have full access to the system hardware and kernel code as well.

We've done a considerable amount of development using Turbo-C and DOS in the past, and it is truly amazing how infrequently we had to reboot Linux during development of the SRA. Back under DOS, we usually had to reboot several times per day. With Linux, we had to reboot only three or four times during the entire development period.

Shared Memory

Figure 10. SRA Memory Usage

Once the RT-Linux programs/modules capture the data, they must be written to storage and displayed for the system operator. We accomplish this by using shared memory. The SRA has 64MB of RAM and we configured the kernel to boot using **mem=61m** which causes the kernel to manage only the lower 61MB, leaving 3MB untouched. It is this 3MB that we use for real-time data capture and as a common communication buffer area between RT-Linux modules and normal user-space programs. Figure 10 depicts the SRA memory usage.

We wrote a single C program (rgc.c) which provides most of the interface between Linux user mode and RT-Linux. This program is a simple command-line style program with tons of commands to read and write data space in common between RT-Linux and user space. Most of our Tcl/Tk scripts merely open a pipe to this program and use it to pass commands and extract data from the system. The program can also be used directly from the command line. This makes development and debugging simpler.

One of the run-line options to **rgc** causes it to loop, testing for data to be written to disk. If no data are ready, the program sleeps for one second. If data are ready, they are extracted and written to the specified disk file.

Linux Laptops

We use up to five laptops on the SRA at once: three for collecting GPS data (one laptop for each GPS receiver) and two for control and display of real-time SRA data. A personal laptop is used for control, and if we're both on the flight, we can both run several instances of the same display programs using another personal laptop. We each have our favorite color-bar for the image of the sea. We'll frequently use one machine to control the SRA and the other to write or modify display or system software as we're flying. The laptops are Chembook 9780s. Each has a 4GB internal hard drive and a modular 6.4GB drive (in place of the floppy), a 14.2'' XGA LCD display, PCMCIA Ethernet card and a 233MHz Pentium-Pro CPU.

Each of these machines dual boots either Red Hat Linux 5.1 or MS Windows 95. To use the laptops as X terminals, we boot Linux, then run the Xfree86 server. We run the X server such that the laptop becomes an X terminal for the SRA data system. This puts most of the burdensome display processing on the laptop processor, since the X server seems to be where the CPU cycles go. There are two ways to cause X to act as an X terminal. The first is:

```
X -query
```

and the second:

```
X -indirect
```

The target machine must be running XDM (X display manager) for this to work. The first method will link directly to the target machine, where you see a typical XDM login prompt. This first method is what we use when controlling the SRA data system. The second method will give you a list of all the machines known on the network to support XDM or X terminals. It is useful back at the lab where many potential hosts are available to pick from.

You can even have two or more X servers running at once. Here's an example:

```
X -query first-machine
X :1 -query second-machine
X :2 -query third-machine
X :3 -query fourth-machine
```

You can get a local X server going with the command:

```
startx -- :4
```

The SRA system configured for storm-surge measurements consists of three Chembook Pentium laptops which dual boot Linux and DOS. The GPS data acquisition program was written for DOS, so each laptop runs this DOS program when collecting the GPS data. After the mission, we reboot the machines to Linux and transfer the data to the SRA data system where it is archived with the other mission data. Once it is all together, we transfer it to the two laptops. In this way, we have triplicated the data. We then take the laptops with us back to the hotel and begin analyzing the data. All five laptops and the SRA data system are on a 10baseT Ethernet network.

GPS and RS-232 Aircraft Data

Some aircraft data are read via RS-232. For this, we are using the standard /dev/ttyS*xx* ports and drivers. The aircraft data are in a 9600 baud stream occurring once per second and the GPS produces a position message twice per second. We use our GPS message to drive a simple Blt plot of the latitude versus longitude, so we can track the progress of the flight.

The RS-232 data are actually captured by the rgc program, since the RT-Linux modules can't make use of the native Linux drivers and we didn't need to rewrite drivers that were working perfectly. Once the data are read, they are copied to the shared memory area above 61MB where any of the programs can access it. Normally, it is accessed by another invocation of rgc and read.

Pitch, Roll, Heading and Track Angle

Accurate aircraft attitude, heading and track angle data are critically important to the SRA in real time. The pitch-and-roll attitude of the aircraft is taken from the on-board Inertial Navigation Units (INU), using Synchro-to-digital converters —one for each parameter. These are read by the RT-Linux module during each scan line. The heading and track information is presently provided via RS-232 from the on-board aircraft data system, which has a direct interface to the INU's digital data stream.

Resulting Data (Radar, GPS, Aircraft)

The SRA radar data are written to disk files by the rgc program. The aircraft data are captured by a separate program and written to a separate disk file. This data is normally captured for the entire duration of the flight, providing a complete flight record in a single file. The carrier-phase GPS data are captured continuously from 45 minutes before the flight until 45 minutes after the flight. The pre- and post-mission data are necessary to resolve the aircraft position to the centimeter level.

System Software Development

Before any Linux development was carried out, we felt it necessary to write some DOS code to work with the Gage digitizer board. Turbo-C version 5.0 was required to compile and use the Gage-supplied library. Once we were successful in getting a Gage example program to work on DOS, we worked with Gage engineers to communicate directly with the digitizer using a normal user-mode program. The main trick was to make the DOS program configure the digitizer and then exit without powering it down. The second trick was to boot from DOS into Linux; this turned out to be quite easy with loadlin.

We determined the PCI board settings for the digitizer by reading /proc/pci and then hard-coding various test programs with the values. We wrote various normal user-mode programs to become familiar with the digitizer. We were able to manipulate the digitizer card in every way except handling interrupts. The **gdb** debugger was a big help throughout the development.

Microcontroller Software Development

A substantial part of the SRA software is actually firmware resident on various microchips.

Microchip provides, at no cost, a very complete and easy-to-use development package for their 16C65A (and other) microcontrollers. It sports a comprehensive simulator, making it possible to watch simulated execution of

quite extensive programs. The only downside is the system runs only on MS Windows.

RT-Linux

The RT-Linux extensions provide just the right features for a real-time data system such as the SRA. The extensions provide much more capability than we actually use in the SRA. We use it to start an RT task at the end of each raster scan. The task processes all the data captured during the previous scan and makes a number of calculations necessary to configure the system for the next raster.

Linux User to RT-Linux Interface

Figure 11. SRA Progam Block Diagram

We wrote rgc.c to be a liaison between normal user processes under Linux and the RT-Linux SRA module. Quite simply, rgc sets up a pointer to the shared memory space that the SRA RT module uses for data storage. They understand each other because they share a common .h file defining the data organization in the shared memory space. Figure 11 depicts how the various SRA real-time programs communicate with each other. **rgc** usually reads commands from stdin and writes to stdout. If it is invoked with certain switches, it forks and polls for RS-232 data and/or writes captured data from the shared memory to disk, all the while taking its commands from stdin. The command set is simple ASCII strings such as **set thresh 24** or **get roll**. The Tcl/Tk programs each open a pipe to their own private rgc, then send commands and receive data back. Everything is done this way except the topographical image display. That program, **creep.c** (because it creeps up the screen), accesses the shared memory directly. The main reason for concentrating everything into rgc is that it generally means we need only recompile rgc, creep and the SRA module when something is added or removed in the shared memory area. In short, it makes for quicker development.

Linux X Terminal—System Display and Control

Figure 12 is a screen shot of the SRA control laptop during hurricane Bonnie's landfall. The image on the left side of the screen is the real-time topographic display. It is gray-scale encoded so that the higher things are, the more white they appear; the lower, the darker. This image clearly shows waves on the left side of the image, the beach in the center and a very distinct dune line. We also have a color-encoded version of this program, but its interpretation is not as intuitive. The blue/brown display represents the attitude of the aircraft. It is a short Tcl/Tk script which reads aircraft attitude data captured by the SRA RT-Linux module.

Figure 12. SRA Screen in Operation during Bonnie.

The bright green display shows how we control and designate the operating conditions for the SRA. At this time, we manually find the return signal using the slider. Once found, we click the “auto” button and the system will keep the ground in the center of our digitizer window, regardless of aircraft altitude variations. The flight map is yet another short Tcl/Tk program. It extracts GPS position data from the shared memory area and uses it to map our position.

TK, Blt, Xview

Tcl 7.6 and Tk 4.2 with Blt 2.3 are used extensively in the SRA. Initially, we thought it might be useful only for prototyping, but it soon became obvious that the X server would be the display bottleneck and not Tcl/Tk.

During development and before we purchased the laptops for control, we used a monitor connected directly to the SRA system. This meant that the X server would run there too. When we began experimenting with using a remote X server, we quickly discovered that the burden of the X server had also moved to the remote system. This was a no-effort way to automatically distribute the load across one or more computers in the system.

We wrote the image display in C using the Xview library. We used this library because we already had a book about it, and it didn't look too difficult to use. It writes each scan line directly to the display and simultaneously to a “pix-map”. When a “repaint” event occurs, the pix-map is used to repaint the whole image. A great way to put a load on the display computer X server is to grab the image map and move it around the screen. The load on the displaying computer will go through the roof, but the data system will remain unaffected.

Data Analysis—Yorick

Once we had some SRA data, we obviously needed to build some software to review it. We wanted to have SRA processing software on several machines and without licensing hassles. That way, we would be able to develop programs at home, on an office laptop (which is also used to control the SRA), on the SRA data system computer and on office Linux and Windows PCs. In total, we needed processing on at least five to ten different systems. We considered IDL, Matlab and Yorick. Our tool of choice for processing was Yorick. It is free, very powerful and will run on a wide variety of platforms including almost every UNIX machine known, Linux and Windows. It has the ability to save data so it can be read on a big-endian or little-endian machine.

Figure 13. Initial Data Product from Yorick Showing Surface Topographic Images Superimposed on NOAA Wind Plots

We first heard of Yorick from an article in *Linux Journal* ("The Yorick Programming Language", Cary O'Brien, July 1998). We downloaded it and gave it a try. We like its C-like syntax and ability to load (and reload) individual functions of a program. This makes for a very powerful and flexible development environment. One of its best features is its cost—free! To put either Matlab or IDL on all the machines would have been prohibitively expensive. Since we have Yorick on the SRA data system and on the controlling laptop, we can easily analyze data in the field with minimal effort using the Linux laptop. Figure 13 shows topographic images from the SRA overlaid on a wind field plot from the August 24th flight. The sea state was above 18 meters (60 feet) on the northern flight line.

Results

We had two or three short test flights on the NASA C-130 aircraft before we had to pack everything up and ship it to MacDill Air Force Base in Tampa, Florida, for installation on the NOAA hurricane hunter. We removed a number of bugs during these test flights, but not all. When we shipped the system, it still would not track properly.

Once we were all installed on the hurricane hunter, we had a 6-hour test flight. This permitted us to work out almost all of the bugs we had seen earlier and a few new ones. We still had a few problems with the tracking code, which would not track reliably.

Bonnie

Figure 14. Flight Track during Bonnie's Landfall

We flew two missions in hurricane Bonnie: the first on August 24 and the second during landfall on August 26, 1998. During our first transit flight from Tampa to the storm, we were able to isolate and correct the tracker bug and everything started working better than expected. Soon after leaving the east coast of Florida, our topographic display of the sea came alive for the first time, showing real sea state. Ocean waves as high as 63 feet were observed in the northeast quadrant of the hurricane on the 24th. Figure 14 shows our August 26 flight track during landfall overlaid on the aircraft weather radar image and a contour plot of the wind field data. The base image includes the weather radar, the wind field and the coastline and was provided by the Hurricane Research Division (HRD) of the NOAA Atlantic Oceanographic and Meteorological Laboratory (AOML) in Miami. We produced this overlay using Yorick.

In addition to hurricane Bonnie, we also flew in Earl and Georges.

Conclusion

Thanks to the reliability of Linux and all of the off-the-shelf real-time data processing programs available in that domain, we were able to put together a state-of-the-art data system on a very tight schedule with a great variety of real-time displays. The displays proved to be of great value both in troubleshooting during development and in real-time geophysical assessment and interpretation during data acquisition. As a result, we were able to document for the first time the spatial variation of the wave field in the vicinity of a hurricane and the spatial and temporal variation of the storm surge associated with hurricanes on landfall.

Resources

C. Wayne Wright (wright@osb.wff.nasa.gov) is a Data Systems Engineer for the NASA Goddard Space Flight Center, Laboratory for Hydrospheric Processes, Observational Sciences Branch, Wallops Island, VA. He is a 1984 graduate of the University of Maryland with a degree in Computer Science. His interests include aviation, amateur radio and computers. Away from work, he and his wife Vicki operate a Linux web server.

Edward J. Walsh (walsh@osb.wff.nasa.gov) is a scientist for the NASA Goddard Space Flight Center, Laboratory for Hydrospheric Processes, Observational Sciences Branch, Wallops Island, VA. He received B.S. and Ph.D. degrees in Electrical Engineering from Northeastern University in 1963 and 1967, respectively. Ed is presently on assignment for NASA at the NOAA Environmental Technology Laboratory in Boulder, Colorado.

Archive Index Issue Table of Contents

Advanced search

University of Toronto WearComp Linux Project

Steve Mann

Issue #58, February 1999

Dr. Mann describes his WearComp (“Wearable Computer”) invention and how it has evolved into the same kind of philosophical basis for self determination and mastery over one's own destiny that is characteristic of the Linux operating system that currently runs on WearComp.

This paper is part one of a two-part series. In this part I will describe a framework for machine intelligence that arises from the existence of human intelligence in the feedback loop of a computational process.

I will also describe the apparatus of the invention that realizes this form of intelligence, beginning with a historical perspective outlining its visual and photographic origins. The apparatus of this invention, called “WearComp”, emphasizes self-determination and personal empowerment.

I also intend to present the material within a philosophical context I call COSHER (Completely Open Source, Headers, Engineering and Research) that also emphasizes self-determination and mastery over one's own destiny.

This “personal empowerment” aspect of my work is what I believe to be a fundamental issue in operating systems such as Linux. It is this aspect that WearComp and Linux have in common, and it is for this reason that Linux is the selected operating system for WearComp.

An important goal of being COSHER is allowing anyone the option of acquiring, and thus advancing, the world's knowledge base.

I will also introduce a construct called “Humanistic Intelligence” (HI). HI is motivated by the philosophy of science, e.g., open peer review and the ability to construct one's own experimental space. HI provides a new synergy between humans and machines that seeks to involve the human rather than having computers emulate human thought or replace humans. Particular goals of HI

are human involvement at the individual level and providing individuals with tools to challenge society's preconceived notions of human-computer relationships. An emphasis in this article is on computational frameworks surrounding “visual intelligence” devices, such as video cameras interfaced to computer systems.

Problem Statement

I begin with a statement of what I believe to be a fundamental problem we face in today's society as it pertains to computers and, in particular, to computer program source code and disclosure. Later, I will suggest what I believe to be solutions to this problem. Linux is one solution, together with an outlook based on science and on self-determination and individual empowerment at the personal level.

A first, fundamental problem is that of software hegemony, seamlessness of thought and the building of computer science upon a foundation of secrecy. Advanced computer systems is an area where a single individual can make a tremendous contribution to the advancement of human knowledge, but is often prevented from doing so by various forms of software fascism. A system that excludes any individual from exploring it fully may prevent that individual from “thinking outside the box” (especially when the box is “welded shut”). Such software hegemonies can prevent some individuals from participating in the culture of computer science and the advancement of the state of the art.

A second fundamental problem pertains to some of the new directions in human-computer interaction (HCI). These new directions are characterized by computers everywhere, constantly monitoring our activities and responding *intelligently*. This is the ubiquitous surveillance paradigm in which keyboards and mice are replaced by cameras and microphones watching us at all times. Perpetrators of this environmental intelligence claim we are being watched for our benefit and that they are making the world a better place for us.

Computers everywhere constantly monitoring our activities and responding intelligently have the potential to make matters worse from the software hegemony perspective, because of the possibility of excluding the individual user from knowledge not only of certain aspects of the computer upon his or her desk, but also of the principle of operation and the function of everyday things. Moreover, the implications of secrecy within the context of these intelligence-gathering functions puts forth a serious threat to personal privacy, solitude and freedom.

Figure 1. Evolution of the WearComp Invention

Computer Science or Computer Secrecy

Science provides us with ever-changing schools of thought, opinions, ideas and the like, while building upon a foundation of verifiable (and sometimes evolving) truth. The foundations, laws and theories of science, although true by assumption, may at any time be called into question as new experimental results unfold. Thus, when doing an experiment, we may begin by making certain assumptions; at any time, these assumptions may be verified.

In particular, a scientific experiment is a form of investigation that leads wherever the evidence may take us. In many cases, the evidence takes us back to questioning the very assumptions and foundations we had previously taken as truth. In some cases, instead of making a new discovery along the lines anticipated by previous scientists, we learn that another previous discovery was false or inaccurate. Sometimes these are the biggest and most important discoveries—things that are found out by accident.

Any scientific system that tries to anticipate "what 99% of the users of our result will need" may be constructing a thought prison for the other 1% of users who are the very people most likely to advance human knowledge. In many ways, the entire user base is in this thought prison, but many would never know it since their own explorations do not take them to the outermost walls of this thought prison.

Thus, a situation in which one or more of the foundation elements are held in secret is contrary to the principles of science. Although many results in science are treated as a "black box", for operational simplicity there is always the possibility that the evidence may want to lead us inside that box.

Imagine, for example, conducting an experiment on a chemical reaction between a proprietary solution "A", mixed with a secret powder "B", brought to a temperature of 212 degrees T. (Top-secret temperature scale which you are not allowed to convert to other units.) It is hard to imagine where one might publish results of such an experiment, except perhaps in the *Journal of Non-Reproducible Results*.

Now, it is quite likely that one could make some new discoveries about the chemical reaction between A and B without knowing what A and B are. One might even be able to complete a doctoral dissertation and obtain a Ph.D. for the study of the reaction between A and B (assuming large enough quantities of A and B were available).

Results in computer science that are based, in part, on undisclosed matters inhibit the ability of the scientist to follow the evidence wherever it may lead. Even in a situation where the evidence does not lead inside one of the secret

“black boxes”, science conducted in this manner is irresponsible in the sense that another scientist in the future may wish to build upon the result and may, in fact, conduct an experiment that leads backwards as well as forwards. Should the new scientist follow evidence that leads backwards, inside one of these secret black boxes, then the first scientist will have created a foundation contaminated by secrecy. In the interest of academic integrity, better science would result if all the foundations upon which it was built were subject to full examination by any scientist who might, at some time in the future, wish to build upon a given discovery.

Thus, although many computer scientists may work at a high level, there would be great merit in a computational foundation open to examination by others, even if the particular scientist using the computational foundation does not wish to examine it. For example, the designer of a high-level numerical algorithm who uses a computer with a fully disclosed operating system (such as Linux) does other scientists a great service, even if he uses it only at the API level and never intends to look at its source code or that of the Linux operating system underneath it.

Figure 2. ECE1766 Class Picture

Obvious or Obfuscated

Imagine a clock designed so that when the cover was lifted off, all the gears would fly out in different directions, such that a young child could not open up his or her parents' clock and determine how it works. Devices made in this manner would not be good for society, in particular for the growth and development of young engineers and scientists with a natural curiosity about the world around them.

As the boundary between software and hardware blurs, devices are becoming more and more difficult to understand. This difficulty arises in part as a result of deliberate obfuscation by product manufacturers. More and more devices contain general-purpose microprocessors, so that their function depends on software. Specificity of function is achieved through specificity of software rather than specificity of physical form. By manufacturing everyday devices in which only executable code is provided, manufacturers have provided a first level of obfuscation. Furthermore, additional obfuscation tools are often used in order to make the executable task image more difficult to understand. These tools include strippers that remove things such as object link names and even tools for building encrypted executables which contain a dynamic decryption function that generates a narrow sliding window of unencrypted executable, so that only a small fragment of the executable is decrypted at any given time. In this way, not only is the end user deprived of source code, but the executable

code itself is encrypted, making it difficult or impossible to look at the code even at the machine-code level.

Moreover, complex programmable logic devices (CPLDs), such as the Alterra 7000 series, often have provisions to permanently destroy the data and address lines leading into a device, so that a single chip device can operate as a finite-state machine yet conceal even its machine-level contents from examination. (See Resources 1 for an excellent tutorial on FPGAs and CPLDs.) Devices such as Clipper chips go a step further by incorporating fluorine atoms, so that if the user attempts to put the device into a milling machine to mill it off layer by layer for examination under an electron microscope, the device will self-destruct in a quite drastic manner. Thus, the Clipper phones could contain a "Trojan horse" or some other kind of back door and we might never be able to determine whether or not this is the case—yet another example of deliberate obfuscation of the operational principles of everyday things.

We have a growing number of general-purpose devices in which the function or purpose depends on software, downloaded code or microcode. Because this code is intellectually encrypted, so is the purpose and function of the device. In this way, manufacturers may provide us with a stated function or purpose, but the actual function or purpose may differ or include extra features of which we are not aware.

Environmental Intelligence Gathering Systems

A number of researchers have been proposing new computer user interfaces based on environmental sensors. Buxton, who did much of the early pioneering research into intelligent environments (smart rooms, etc.), was inspired by automatic flush urinals (as described, for example, in U.S. Pat. 4309781, 5170514, etc.) and formulated, designed and built a human-computer interaction system called the "Reactive Room" (see Resources 2 and 3). This system consisted of various sensors, including optical sensors (such as video cameras) and processing, so that the room would respond to the user's movement and activity.

Increasingly, we are witnessing the emergence of intelligent highways, smart rooms, smart floors, smart ceilings, smart toilets, smart elevators, smart light switches, etc. However, a typical attribute of these "smart spaces" is that they were designed by someone other than the occupant. Thus, the end user of the space often does not have a full disclosure of the operational characteristics of the sensory apparatus and the flow of intelligence data from the sensory apparatus.

In addition to the intellectual encryption described in the previous section, where manufacturers could make it difficult, or perhaps impossible, for the end

user to disassemble such sensory units in order to determine their actual function. There is also the growth of hidden intelligence, in which the user may not even be aware of the sensory apparatus. For example, U.S. Pat. 4309781 (for a urinal flushing device) describes:

> ... sensor... hidden from view and thus discourage tampering with the sensor... when the body moves away from the viewing area... located such that an adult user of average height will not see it... sensing means, will be behind other components... positioned below the solenoid to allow light in and out. But the solenoid acts in the nature of a hood or canopy to shield the sensing means from the normal line of sight of most users.... Thus most users will not be aware of the sensing means. This will aid in discouraging tampering with the sensing means. A possible alternate arrangement would be to place the sensing means below and behind the inlet pipe.

U.S. Pat. 4998673 describes a viewing window concealed inside the nozzle of a shower head, where a fiber optics system is disclosed as a means of making the sensor remote. The concealment is to prevent users from being aware of its presence. U.S. Pat. 5199639 describes a more advanced system where the beam pattern of the nozzle is adapted to one or more characteristics of the user, while U.S. Pat. 3576277 discloses a similar system based on an array of sensing elements.

A method of creating viewing windows to observe the occupants of a space while at the same time making it difficult for the occupants to know if and when they are observed is proposed in U.S. Pat. 4225881 and U.S. Pat. 5726706.

In addition to concealing the sensory apparatus, a goal of many visual observation systems is to serve the needs of the system architect rather than the occupants. For example, U.S. Pat. 5202666 discloses a system for monitoring employees within a restroom environment, in order to enforce hygiene (washing of hands after using the toilet).

Other forms of intelligence, such as intelligent highways, often have additional unfortunate uses beyond those purported by the installers of the systems. For example, traffic-monitoring cameras were used to round up, detain and execute peaceful protesters in China's Tiananmen Square.

U.S. Pat. 4614968 discloses a system where a video camera is used to detect smoke by virtue of the fact that smoke reduces the contrast of a fixed pattern opposite the video camera. However, the patent notes that the camera can also be used for other functions such as visual surveillance of an area, since only one segment or line of the camera is needed for smoke detection. Again, the

camera may thus be justified for one use; additional uses, not disclosed to occupants of the space, may then evolve. U.S. Pat. 5061977 and 4924416 disclose the use of video cameras to monitor crowds and automatically control lighting in response to the absorption of light by the crowds. While this form of environmental intelligence is purportedly for the benefit of the occupants (to provide them with improved lighting), there are obvious other uses.

U.S. Pat. 5387768 discloses the use of visual inspection of users in and around an automated elevator. Again, these provide simple examples of environmental intelligence in which there are other uses, such as security and surveillance. Although even those other uses (security and surveillance) are purportedly for the benefits of the occupants, and it is often even argued that concealing operational aspects of the system from the occupants is also for their benefit, it is an object of this paper to challenge these assumptions and provide an alternate form of intelligence.

When the operational characteristics, function, data flow and even the very existence of sensory apparatus is concealed from the end user, such as behind the grille of a smoke detector, environmental intelligence does not necessarily represent the best form of human-machine relationship for all concerned. Even when the sensors are visible, there must be the constant question as to whether or not the interests of the occupant are identical to those who control the intelligence-gathering infrastructure.

The need for personal space, free from monitoring, has also been recognized (see Resources 4) as essential to a healthy life. As more and more personal space is stolen from us, we may need to be the architects of alternate spaces of our own.

Solution to Software Fascism

The first solution to these problems is a framework called Completely Open Source, Headers, Engineering, and Research (COSHER). Before investing considerable time in learning how to use new software and in developing works for that new software, which may then become *locked* into a particular file format, we ask ourselves a very simple question: is the software in question COSHER?

This means that there has been no deliberate attempt at obfuscation of the underlying principles of the operation of this software or in preventing us from freely distributing the intellectual foundations upon which we may invest many years of our lives. Deliberate attempts at obfuscation include such practices as eliminating source code and stripping executable task images.

By using COSHER software, we are making a statement that we prefer Computer Science to Computer Secrecy. Science supports the basic principles of peer review, a continued development and advancement of software principles and principles that we build on top of the software.

Moreover, the time we invest in learning the software as well as creating works in the software will be less likely to go to waste if we have a copy of the complete source code of the software. In this manner, should the software ever become discontinued or unsupported, we will be able to become our own software support group and migrate the software forward to new architectures as our old computers become obsolete. If it is COSHER, chances are we will be less likely to lose the many hours or years we invest in producing works within the software. Furthermore, if we make new discoveries that are built on a foundation of COSHER software, they are easier to distribute.

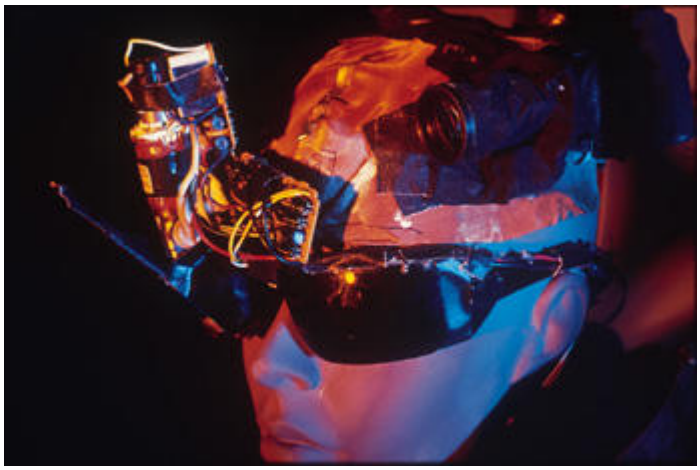
In science, it is important that others be able to reproduce our results. Imagine what it would be like if we had built our results on top of DOS 3.1. Others would have to either rewrite our software to exactly reproduce our results, or find an old version of DOS 3.1. Since this is proprietary software, we are not at liberty to freely distribute it with our research, but it is also no longer available for purchase. However, if we had built our work on COSHER software such as Linux 1.13, we can include a full distribution of Linux 1.13 in an archive together with our results. Many years in the future, a scientist wishing to reproduce our results could then obtain a virtual machine (emulator for our specific architecture which will no doubt be obsolete by then) and install the COSHER operating system (Linux 1.13) that came with our archive, then compile and run our programs.

The Linux operating system is a good example of a COSHER operating system. GNU software is also COSHER. Many COSHER software packages are available, including GIMP (Gnu Image Manipulation Program) and the VideoOrbits software package (described in http://wearcam.org/orbits/index.html).

Solution to Environmental Intelligence Gathering

I propose a computational framework for individual personal empowerment. This framework is based on my “WearComp” invention—an apparatus for (embodiment of) realization of HI.

This framework involves designing a new kind of personal space. An embodiment of the “WearComp” invention is an apparatus that is owned, operated and controlled by the occupant of that space. In one sense, the apparatus of this invention is like a building built for one occupant and collapsed down around that one occupant.



WearComp as a Basis for HI

I invented WearComp in Canada in the 1970s as a photographic tool for the visual arts (see Resources 5), in particular, something I called “mediated reality” (altered perception of visual reality). The goal of mediated reality, unlike related concepts such as virtual (or augmented) reality, was to reconfigure (augment, deliberately diminish or otherwise alter) the perception of reality in order to attain a heightened awareness of how ordinary, everyday objects respond to light.

HI is a new form of human-computer interaction comprising a computer that is subsumed into the personal space of the user (e.g., the computer may be worn, hence the term “user” and “wearer” of the computer are interchangeable), controlled by the wearer, with both operational and interactional constancy (e.g., it is always on and always ready and accessible [see Resources 6]).

The WearComp invention, described in IEEE Computer, Vol. 30, No. 2 at http://wearcomp.org/ieeecomputer.htm (a historical account was given in IEEE ISWC-97, October 1997 and is also on-line at http://wearcomp.org/historical/index.html) forms the basis for HI. The evolution of the apparatus of this invention is depicted in Figure 1.

Definition of WearComp

A wearable computer is a computer that is subsumed into the personal space of the user, controlled by the user and has both operational and interactional constancy.

Most notably, it is a device that is always with the user and into which the user can always enter commands and execute a set of entered commands while walking around or doing other activities.

The most salient aspect of computers in general (whether wearable or not) is their *reconfigurability* and their *generality*, e.g., their function can be made to

vary widely, depending on the instructions provided for program execution. This is true for the wearable computer (WearComp). For example, the wearable computer is more than just a wristwatch or regular eyeglasses; it has the full functionality of a computer system and, in addition, is inextricably intertwined with the wearer.

This is what sets the wearable computer apart from other wearable devices such as wristwatches, regular eyeglasses, wearable radios, etc. Unlike these other wearable devices that are not programmable (reconfigurable), the wearable computer is as reconfigurable as the familiar desktop or mainframe computer.

The formal definition of wearable computing defined in terms of its three basic modes of operation and its six fundamental attributes is provided elsewhere in the literature. (See Resources 7.)

WearComp, as Universal Interface to Reality

Such a computational framework allows one to subsume all of the personal electronics devices one might normally carry, such as cellular phone, pager, wrist watch, heart monitor, camera and video camera into a single device. Obviously, since it is a fully featured computer, it is possible to respond to e-mail, plan events on a calendar, type a report, etc., while walking, standing in line at the bank or anywhere. In this way, WearComp anticipated the later arrival of the so-called “laptop computer”, but has advantages over the laptop in the sense that it can be used while walking around doing other things. However, the real power of WearComp is in its ability to serve as a basis for personal imaging and humanistic intelligence.

Figure 3. Another Example of WearComp

Personal Safety Device

WearComp not only subsumes the function of the laptop computer, but goes beyond it. Another area in which WearComp provides a truly new form of user interface not found on laptops and PDAs (personal digital assistants) is in its constancy of user interface and operation. This characteristic may become most evident in its use as a personal security camera. Imagine, perhaps as you walk down some quiet street at night, an assailant appears, demanding cash from you. You would not likely have the time or opportunity to pull out a camcorder to record the experience, but since the eyeglasses are worn constantly, you would have a video record of the experience to aid investigation.

Camera of the Future

Less extreme examples of WearComp as a new user-interface include the ability to construct a personal documentary video without conscious thought or effort. For example, in a fully mediated reality, all light entering the eyes, in effect, passes through the computer and may therefore be recorded (and possibly transmitted to remote locations). Wearable Wireless Webcam (see Resources 8) is an example of a personal documentary video recorded using a reality mediator.

In the future, we may very well have the capability to capture and recall our own personal experiences and to have photo albums generated automatically for us. We will never miss baby's first steps, because we will have a retroactive record feature that lets us, for example, “begin recording from 5 minutes ago”. Photo albums, in addition to being generated automatically, may also be exhibited while they are being generated. Rather than sending postcards to friends and relatives or showing them an album after you come back from vacation, you may just put on your sunglasses and have the album sent to them automatically, as was done with the Wearable Wireless Webcam experiment in which video was transmitted and still images automatically selected from the video.

Personal Intelligence Arms Race

While there will no doubt be more environmental intelligence than personal intelligence, there is at least the hope that there might be an end to the drastic imbalance between the two. The individual making a purchase in a department store may have several cameras pointing at him to make sure that if he removed merchandise without payment, there would be evidence of the theft. However, in the future, he will have a means of collecting evidence that he did pay for the item, or a recorded statement from a clerk about the refund policy. More extreme examples such as the case of Latasha Harlins, a customer falsely accused of shoplifting and fatally shot in the back by a shopkeeper as she attempted to walk out of the shop, come to mind.

In this sense, the camera-based reality mediator becomes an equalizer much like the Colt 45 in the “Wild West”. In the WearCam case, it is simply a matter of mutually assured accountability.

Future Directions

Much work remains to be done in development of this project. Currently, I teach Electrical and Computer Engineering (ECE1766) at the University of Toronto. To the best of my knowledge, this is the world's first course on how to be a “cyborg” entity. Students learn not only by doing, but by *being*. I call this

form of learning *existential learning*. Each student creates a “reconfigured self”--a new form of personal space. Thus, students learn about the concept of personal empowerment from a first-person perspective through personal involvement.

We are writing new protocols for the altered perception of reality (mediated reality) that the WearComp provides. One example is picture-transfer protocol (PTP), in which packets of variable length are transmitted. Each packet is a JPEG compressed picture. Because of image compression, the amount of data varies depending on image content, hence the packet length depends on image content.

The reason for one packet per picture is that pictures are taken 60 times per second, which is much faster than they can be sent. Thus, whenever there is a lost packet and a re-transmission is needed, a newer picture will most likely be available to be sent instead. With PTP, retransmissions are always current.

Next month I will describe a mathematical (computational) framework called “Mediated Reality”, in which we will see that picture data is of greatest value only if it is up-to-date. Old pictures are of less value when trying to construct a computer-mediated reality. Thus, packet resends should always be of the most current image; hence the design of PTP is based on variable packet lengths, in which the packet length is the length of a picture.

Further information about the WearComp Linux project may be found in http://wearcam.org/ece1766.html.

Resources

Thanks to Kodak and Digital Equipment Corporation (DEC) for assistance with the Personal Imaging and Humanistic Intelligence projects.

Steve Mann, inventor of WearComp (wearable computer) and WearCam (eye-tap camera and reality mediator), is a faculty member at the University of Toronto, Department of Electrical and Computer Engineering. Dr. Mann has been working on his WearComp invention for more than 20 years, dating back

to his high school days in the 1970s. He brought his inventions and ideas to the Massachusetts Institute of Technology in 1991, founding what later became the MIT Wearable Computing Project, and received his Ph.D. from MIT in 1997 in this new field he had established. Anyone interested in joining or helping out with the “community of cyborgs” project or the WearComp Linux project may contact the author by e-mail at mann@eecg.toronto.edu.

Archive Index Issue Table of Contents

Advanced search

Virtual Network Computing

Brian Harvey

Issue #58, February 1999

Mr. Harvey tells us about the VNC software package and how to set it up to control MS Windows servers from Linux.

In today's changing world, an increasing number of UNIX system administrators are finding they need to support Windows NT servers in their work environments. Whether Exchange or application servers, NT servers are starting to creep into what were once UNIX-only shops. The responsibility of managing an NT server can be discouraging to any UNIX guru. UNIX users are used to the flexibility of the X Window System—the ability to run applications easily on any UNIX server and have remote X applications display on the local desktop. It is much more difficult to manage NT servers remotely and the administrator usually needs to be at the system's console to run most NT applications.

Several commercial products allow MS Windows applications to be controlled remotely from an X desktop. In addition, there are several commercial X servers for MS Windows which allow the opposite. However, until recently an equivalent free software package was not available.

VNC

Researchers at the Olivetti & Oracle Research Laboratory (ORL) have released the VNC software package under the GNU general public license. VNC, which stands for Virtual Network Computing, is a client/server-based, stateless, platform-independent protocol developed at ORL. This protocol implements a remote display system in which a user is allowed to control a computing

“desktop” managed by a VNC server, by connecting to it from a VNC client application called a “viewer”. VNC servers currently exist for Windows 95/NT, Macintosh and UNIX. A variety of VNC clients exist as well for a number of operating systems. Figure 1 shows all the connections currently possible with the VNC protocol. Many of the VNC viewers are ported by users on the Net. ORL supplies precompiled server and client binaries for Windows 95/NT, Macintosh, Linux, Digital UNIX and Solaris. In addition, ORL provides a Windows CE client.

Figure 1. VNC Protocol Connections

In this article, I will discuss how to set up the VNC software, allowing you to control a Windows desktop from Linux running the X Window System (probably the most common use of VNC for Linux users).

Installation

Linux (VNC Viewer)

ORL provides an x86 Linux 2.0 binary that works great with Red Hat 5.1 and can be retrieved from their download page (see Resources). Once you have the package, unarchive it using **gunzip** and **tar**. The binary distribution provides no installation script, but for our purposes we simply need to have root copy the viewer binary, **vncviewer**, into a suitable location accessible by others, such as /usr/local/bin.

Windows 95/NT (VNC Server)

ORL provides a precompiled Windows 95/NT binary supplied as a package that can also be downloaded from their download page. The package installs like most other Windows software packages, i.e., using InstallShield. The VNC server (WinVNC) can be installed as a regular application (started/stopped by the user currently logged on to the console) or as an NT service (starts automatically when NT boots; does not exit when user logs out). Installation as a service is a new feature in recent versions of WinVNC. The latest version at the time of writing, is 3.3.2R5. VNC is actively developed, so a newer version of the software will most likely be ready to download by the time you read this. I recommend installing WinVNC as a service so that the VNC server is always running and you do not have to remain logged in on the Windows console at all times.

To install WinVNC as a service, simply install the package as you would normally install any other Windows application, then type in a command window:

```
cd
WinVNC.exe -install
WinVNC.exe -run # or reboot NT to have the<\n>
                # service start automatically
```

Configuration

The Linux VNC viewer requires no configuration to use. However, the Windows VNC server does require some minor configuration. To bring up the configuration window, either right-click on the WinVNC icon in the Windows NT/95 system tray and select **Properties**, or open a DOS command window and type:

```
cd
WinVNC.exe -settings
```

Figure 2. Windows VNC Configuration Window

In the configuration window, shown in Figure 2, the following options can be set:

- Make sure **Accept Socket Connections** is selected. If this option is not checked, all incoming connections will be disabled.
- The **Display Number** can be left at 0. This value is specified when using a VNC viewer to connect to this server.
- Set a **Password** to secure access to this VNC desktop (a good idea). When connecting to this VNC server via a viewer, you will be prompted for the same password.
- If **Disable Remote Keyboard & Pointer** is selected, all incoming viewer connections will be able to see the desktop but will not be able to move the mouse or type anything (a read-only connection).
- In the **Update Handling** section, various options can be turned on/off to control how the VNC server sends "desktop changes" to a VNC viewer. See http://www.orl.co.uk/vnc/winvnc.html for in-depth explanations on the pros and cons of each option.

Press the **Ok** or **Apply** button to apply your configuration changes.

Using the Linux VNC viewer

Once you have WinVNC running on a Windows server, try connecting to it from your Linux desktop by typing (within X) the following command, followed by the password you gave when configuring WinVNC (if any):

```
> vncviewer
vncviewer: VNC server supports protocol version 3.3 (viewer 3.3)
Password:
vncviewer: VNC authentication succeeded
vncviewer: Desktop name "boxster"
vncviewer: Connected to VNC server, using protocol version 3.3
vncviewer: VNC server default format:
16 bits per pixel.
Least significant byte first in each pixel.
True color: max red 31 green 63 blue 31
   shift red 11 green 5 blue 0
```

```
Using default colormap and translating to BGR233
Creating window depth 8, visualid 0x22 colormap 0x21
```

If you typed the password correctly, several lines of information will appear and a new large window will pop up showing the entire remote Windows desktop. When you are finished using the VNC viewer, simply close the viewer's window to close the connection. The remote Windows desktop will be left in the last state the viewer left it in.

Figure 3 shows a sample Linux desktop with a newly opened VNC viewer connection “viewing” a Windows NT desktop.

Figure 3. Linux Desktop Viewing Windows NT

A nice feature available in recent VNC releases is the ability to send the infamous **ctrl-alt-del** key sequence to the Windows desktop shown in a VNC viewer. This feature has distinct advantages when the VNC server is installed as a service:

- If the VNC server is installed as a service under Windows NT, you don't need to have a user logged on all the time with the VNC server running as a Windows application. When it comes time to use that server remotely, simply connect to it with a VNC viewer, press **ctrl-alt-del** to get the NT login Window, and log on as you normally would to the NT box.
- If you need to stay logged on to the NT server but want to exit your local X session, you can type **ctrl-alt-del** to get the “Windows NT Security” pop-up window, click on “Lock Workstation” to lock the console, close the VNC viewer connection, then exit your X session. You will still remain logged on to the NT server; its screen is now locked.

Advantages

The VNC protocol has several advantages. The main one is that it is stateless. A user can close a connection to a remote desktop from one VNC viewer and later reconnect to that same remote desktop from the same or different VNC viewer, and it will be in the same state.

When using the Java VNC viewer, a system administrator can control a Windows 95/NT, Macintosh, or UNIX desktop from anywhere in the world using a Java-enabled browser. The VNC server can be configured so that all incoming viewer connections will be able to see the desktop but will not be able to move the mouse or type anything (a read-only connection). This option comes in handy in a teaching environment, where each student in a class connects to the instructor's “desktop” and watches a demonstration on his own computer rather than on an overhead connected to the instructor's computer.

How I Use VNC

At work, I have an Alpha running Digital UNIX and a P133 running Windows NT 4.0. Although I am strictly a UNIX systems administrator, my company's e-mail standard is based on Microsoft Exchange. Therefore, I am required to have a Windows desktop on my desk in order to read Exchange e-mail. However, at home I run only Linux. I was looking for a way to read my Exchange e-mail from home. After reading about VNC, I knew I had found what I was looking for.

I use the Linux VNC viewer at home to connect to the Windows NT box on my desk at work over a PPP connection. Figure 4 shows me reading my Exchange e-mail with such a setup. While VNC performance over a PPP line isn't spectacular, it is very usable and solves my problem of not being able to read Exchange e-mail from home.

Figure 4. Reading MS Exchange from Linux

Resources

Brian Harvey is currently a UNIX Systems Administrator for U.S. Technical Services in Huntington Beach, CA. He is a graduate of UC Riverside with a BS and MS in Computer Science. He can be reached via email at brian.harvey@ustsvs.com

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Configuring ATM Networks

Wayne J. Salamon

Issue #58, February 1999

This article describes how to configure Linux-based PCs and an asynchronous transfer mode (ATM) switch to build on ATM network.

The Linux ATM software (device driver and utilities) is developed and supported by Werner Almsberger in Switzerland as part of the Linux-ATM API software set (see Resources). This software contains device drivers for the following ATM adapters: Efficient ENI-155P, SMC ATM Power 155, Rolf Fiedler's TNETA1570 board, Zeitnet ZN1221/ZN1225 and the IDT 77901/77903 155 and 25 Mbps adapters. Also, a driver for the Fore PCA-200E ATM adapter is available separately (see Resources). The two adapters I have experience with are the Efficient ENI-155p and the Fore PCA-200E.

The National Institute of Standards and Technology (NIST) uses ATM and Fast-Ethernet networks as interconnects in its scalable cluster computing initiative. One research area is evaluating the benefits of ATM and Fast-Ethernet networks in this cluster environment.

In this article, I will tell you how to obtain and install the ATM support software and device drivers. I will also describe how to configure the ATM connections on the PCs and the switch to be used for IP network traffic.

The ATM interface cards I use are ENI-155P ATM adapters produced by Efficient Networks and PCA-200EPC adapters from Fore Systems. These cards are installed in standard Pentium or Pentium-Pro-based PCs running Linux. The ATM switch I used for this article is a Fore ASX-1000, although the information I give applies to all of the Fore ATM switches. This switch can be set up to allow the Linux workstations to use IP over both Switched Virtual Circuits (SVC) and Permanent Virtual Circuits (PVC).

Obtaining and Installing the Linux-ATM Software

The ATM software is available from http://lrcwww.epfl.ch/linux-atm/. The software is packaged as a compressed, gzipped tar file. Each version of the software is tied to a specific version of the Linux kernel. For this article, I used version 0.35 running on Linux kernel 2.1.90. The size of the ATM software distribution is roughly 500KB. The device driver for the Fore PCA-200E adapter can be obtained by anonymous FTP from ftp://os.inf.tu-dresden.de/pub/pca200e/. Refer to the README file in the PCA200 distribution for further information.

The driver portion of the Linux-ATM software, as well as the changes to the Linux kernel, are shipped as one large patch file. Therefore, adding support to the Linux kernel for ATM is straightforward: apply the kernel patch, configure and rebuild the kernel in the usual way. The ATM configuration items you must have are:

- Asynchronous Transfer Mode (ATM) (CONFIG_ATM)
- Classical IP over ATM with ATMARP (CONFIG_ATM_ATMARP)
- Device driver, one of the following:Efficient Networks ENI155P (CONFIG_ATM_ENI)ZeitNet ZN1221/ZN1225 (CONFIG_ATM_ZATM)Rolfs TI TNETA1570 (CONFIG_ATM_TNETA1570)IDT 77201 (NICSTAR) (CONFIG_ATM_NICSTAR)

I recommend starting with a fresh Linux kernel source tree before applying the ATM patch. Refer to the USAGE file that is part of the Linux-ATM software, as things may change. All of the device drivers in the distribution can be built as kernel modules or as part of the kernel object itself. If you are using a Fore PCA-200E adapter, you do not select a driver during the kernel configuration. The PCA-200E device driver is built as a module separately, as specified in the README file included in the PCA200 distribution.

After the kernel is patched, rebuilt and installed, you are ready to build the ATM support software. Again, refer to the instructions in the USAGE file. One change I recommend is installing the support files in /usr/local/atm-version/bin and creating a soft link from /usr/local/atm to the actual install directory. By using the soft link, you can change ATM software levels and back them out, if needed, without changing the configuration scripts.

Configuring the ATM Device Interface

You are now ready to configure the IP over ATM. First, you must decide what type of “virtual circuits” to use to connect the machines. ATM is a point-to-point, switched technology; in order for two hosts to communicate, a virtual circuit must be established between them.

Switched Virtual Circuits (SVCs) are connections that are established dynamically and torn down when the connection is no longer needed. However, a high latency is associated with establishing a connection. Also, SVCs are deleted after a timeout period if no traffic is sent over the connection. Therefore, the latency associated with SVCs is not always predictable. I encountered several problems when using SVCs, such as connections not being established or sometimes failing to remain open.

Permanent Virtual Circuits (PVCs) are established and kept open. Thus, no latency is associated with establishing the connection, as there is when using SVCs. The disadvantage of PVCs is that the switch must be configured to establish all the connections between the hosts. When you have several hosts and each host needs to communicate with all the others, the number of PVCs required within the ATM switch grows rapidly. Specific configuration information for SVCs and PVCs is discussed later, but I will jump ahead a bit in order to complete the IP configuration now. The steps to configure the ATM interface are as follows:

- Start the ATM software daemons with these commands:

```
atmsigd -b
ilmid -b
atmarpd -b
```

- Create the ATM device name:

```
atmarp -c atm0
```

- Configure the ATM interface for IP:

```
ifconfig atm0 ipaddr netmask netmask mtu mtu
```

- Add the route for the ATM subnet:

```
route add -net network netmask netmask atm0
```

- Create a permanent ATM ARP (address resolution protocol) cache entry for the ARP server:

```
atmarp -s arpserver arpsrvnsap arpsrv
```

ipaddr is the IP address of the ATM interface, **netmask** is the network mask and **network** is the IP address of the network to which we are connecting. **arpserver** is the IP address of the ATM ARP server and **arpsrvnsap** is the ATM address of the ARP server. The ATM ARP server is used to convert an IP address to an ATM network service access point (NSAP) address. (The NSAP address is similar to a media access control (MAC) address and is 20 octets long.) The NSAP address is needed to establish SVCs between nodes. You can also create an /etc/hosts.atm file to contain the IP to NSAP mapping, allowing for quicker IP to NSAP translations. For my network, I use the Fore switch as the ARP server. The **atmarpd** daemon maintains a cache of IP to NSAP mappings. The **atmarp** command makes the ARP cache entry permanent when the **arpsrv** option is used.

One final note: if you are going to use PVCs only, you do not need to start the **atmsigd** and **ilmid** daemons. Listing 1 contains a complete example of configuration commands.

Configuration of Switched Virtual Circuits

The ATM switch configuration commands I use apply to the entire family of Fore ATM switches, because they all have a similar command interface.

When using SVCs, a host must pass information to the ATM switch, declaring its intent to set up a connection with another host. The term for connection setup is “signaling”. The ATM protocol used between a host and a switch is the user-network-interface (UNI) signaling standard. There are several revisions of the UNI standard. The Fore ATM switch supports UNI 3.0, 3.1 and 4.0. The Linux-ATM software also supports these versions.

However, there are standards and there are implementations. In setting up our SVCs, I encountered several problems with UNI 3.0 signaling. The UNI 3.1 signaling was more stable and reliable. To change the signaling on the Fore ATM switch, each port must be changed individually, using the switch control processor (SCP) command interface.

First, log on to the SCP via a TELNET session or by using a terminal attached to the serial port on the Fore switch. The command syntax used here is the same as Fore's. Required parameters are shown between “<” and “>”; options are enclosed in brackets (“[” and “]”); modifiers to options are enclosed in parentheses. One of the modifiers must be chosen.

Change to the UNI configuration menu:

```
localhost::> conf uni
```

The switch prompt is shown in italics, while the command is shown in normal text.

The command **show** will list the current UNI status for each port. If the port is already configured for UNI 3.1, no change needs to be made. Otherwise, you must first delete the current configuration. The syntax for the delete command is **del *port vpi***, where *port* is the switch port and *vpi* is the virtual path identifier (usually 0). To delete the signaling on port 1A1 for VPI 0, you would enter this command:

```
localhost::configuration uni> del 1a1 0
```

Now you are ready to configure the port for UNI 3.1. The syntax for the new command is:

```
new <port><vpi> [auto | uni30 | uni31] [-ilmi (up | down)]
```

The **ilmi** option is used when you want the port to respond to integrated local management interface (ILMI) requests. ILMI is used by the hosts to obtain the ATM NSAP address assigned to the host. You usually want to have ILMI active for the port, so the command for port 1A1, VPI 0 is:

```
localhost::configuration uni> new 1a1 0 uni31 -ilmi up
```

Now that the ATM switch ports have been configured, the software on the workstation must be set up. The key portions of the Linux-ATM software are three daemons: **atmsigd** to handle signaling (UNI), **ilmid** to handle ATM address registration and **atmarpd** to map ATM addresses to IP addresses. Listing 1 is the startup script I use to start the ATM daemons and to configure the ATM interface on a host. This script can be called from a system startup script (/etc/rc.d/rc.local, for example) to configure the ATM interface at boot time.

The ATM signaling daemon, **atmsigd**, must be compiled specifically for the version of signaling you wish to use and must be compatible with the signaling version the ATM switch port has been configured to use. The default version used in the ATM software is UNI 3.0. If you've configured the switch to use UNI 3.1, having the hosts use UNI 3.0 will most likely work, due to backward compatibility. However, I recommend you configure the Linux-ATM software to use the same version as the switch, UNI 3.1.

To have the signaling daemon use UNI 3.1, edit the Rules.make file in the ATM source directory (/usr/src/atm if you follow the steps in the USAGE file). You need to change the STANDARDS line to specify the version of signaling to support. For UNI 3.1, this line should be: **STANDARDS=-DUNI31**.

Special Configuration for ENI-155p ATM Cards

If you are using Efficient ENI-155p ATM cards, the number of simultaneous virtual channels available is limited. The ENI card performs the segmentation and reassembly (SAR) of ATM cells by using memory on the adapter card as a buffer. The host ATM software allocates buffer space for each virtual channel. If you attempt to open more SVCs than are supported by the available buffer space, you will receive this error message from the ATM ARP daemon:

```
atmarpd:IO: [2]connect: No buffer space available
```

When IP over ATM is used, the device driver sends packets to the ATM card using an ATM Adaptation Layer (AAL). While several adaptation layers are available, AAL-5 is used for IP. The AAL-5 packet is a type of service data unit (SDU) and is somewhat analogous to an Ethernet frame. The AAL-5 packets are divided into individual ATM cells by the Efficient ATM adapter.

The MTU (maximum transmission unit) size for the ATM interface depends on the SDU size. The IP over ATM (Classical IP) specification says that the MTU should be no larger than 9180 bytes. There are also 8 bytes for an AAL-5 trailer, so the SDU for IP over ATM is 9188 bytes in the default configuration. The amount of buffer space needed on the card depends on the maximum SDU size.

The Linux-ATM software allocates three times the maximum SDU size, rounded up to the nearest power of two. In the default configuration, this allocation results in 32KB of buffer space being reserved for each ATM connection (9180 x 3 = 27540, rounded to 32768 bytes). Also, using classical IP causes two SVCs to be made: the initiating machine opens an active connection to the target machine and the target machine opens an active connection back, that is, a passive connection on the initiator. Therefore, these two connections result in the allocation of two buffers on the ATM card, for a total of 64KB.

The default configuration allows a host to have a maximum of fourteen simultaneous connections when using the “client” version of the ENI-155p ATM card, which has 512KB of memory and 504KB of memory available for the SAR buffers. These fourteen connections allow communication using IP over ATM to seven other hosts when using SVCs. If you set up PVCs, you can communicate with fourteen other hosts. When using an ARP server, you have one less connection available, reducing the host count by one as well. The “server” version of the ENI 155p card has 2MB of memory, with 2040KB for SAR buffers, allowing for more simultaneous connections.

To increase the number of simultaneous connections for classical IP, you need to change the size of the maximum SDU set on the ATM interface. By using the allocation rule given above, you can estimate the amount of memory needed for the connections. For example, if you want to use 16KB for each connection, the maximum SDU would be 16384 divided by 3, which is 5461 bytes. I'll use an SDU of 4352 bytes for my example in this article.

The maximum SDU is specified as an option to the ATM ARP daemon. However, when the SDU is changed, the IP interface must also be configured to have an MTU of the same size as the SDU, minus 8 bytes for the AAL-5 trailer. Therefore, in my example the MTU is 4344 bytes.

A potential problem occurs when changing the maximum SDU for the interface: the ATM ARP daemon (**atmarpd**) may not communicate with the ARP server on the Fore switch. Our switch would accept only connections with an SDU of 9188 bytes. The fix for this problem is to create a permanent ARP cache entry on the host, specifying the maximum SDU of 9188 bytes, for the connection to the ARP

server. The steps for configuring the ATM software on the workstation are as follows:

- Configure the IP interface for your MTU size, 4344 bytes in my example:

```
ifconfig atm0 ipaddr netmask netmask mtu 4344
```

- Create a permanent ATM ARP cache entry for the ARP server with SDU size of 9188:

```
atmarp -s arpserver arpsrvnsap qos \
ubr:sdu=9188 arpsrv
```

- Configure the SDU (MTU plus 8 bytes) on the ATM interface:

```
atmarp -q network ubr:sdu=4352
```

Refer to Listing 1 for a complete example of configuring the ATM software for the Efficient adapter.

Using IP over Permanent Virtual Circuits

To establish a PVC, the following steps must be performed.

- On the workstation, add an ATM ARP entry on each node specifying the PVC (*vpi.vci* pair) used to connect to each of the other hosts.
- Create the PVC on the switch.

As an example, the following commands executed on the appropriate host will set up a PVC between nodes named node1 and node2, on interface 0, using a vpi of 0 and a vci of 70:

- ***node1 :*** **atmarp -s node2 0.0.70**
- ***node2 :*** **atmarp -s node1 0.0.70**

The PVC is identified by three numbers, separated by two periods. The numbering scheme is *interface.vpi.vci*, where interface is 0 for the first ATM adapter, 1 for the next, etc. The default interface for the atmarp command is 0. The *vpi* (virtual path identifier) and *vci* (virtual channel identifier) are the standard ATM PVC identifiers. The host name (node1 and node2) can be used if there is an entry for it in the /etc/hosts file; otherwise, use the IP address of the host.

The commands above tell node1 to communicate with node2 over PVC **0.0.70** and for node2 to communicate with node1 over PVC **0.0.70**. The atmarp command links the IP address of the target host to the PVC. You could choose a different PVC for each connection, but it is simpler to think in terms of one PVC connecting two machines.

The *vpi.vci* pair must not be in use on the host. Also, any ATM ARP cache entries must be deleted for the target host before creating the PVC. (These cache entries are created when SVCs are opened to the destination host.) To delete an ARP cache entry on node1 for node2, you would use this command:

```
node1: atmarp -d node2
```

Next, the switch must be configured to complete the PVC between the hosts. It is helpful to understand the port naming convention used by the Fore switch. The port names consist of three identifiers:

- Board: the number of the switch board (same as the SCP number); each SCP controls one switch board.
- Network Module: the slot (A, B, C, or D) in the switch board containing the port.
- Port: the physical port number on the network module.

For example, port 1b3 refers to the first switch board, the second network module (module b) on board 1 and the third physical port on the second network module. The Fore ASX-200 switch has only one switch board, while the ASX-1000 switch has four. There is a maximum of four network modules per switch board and a maximum of six physical ports per module.

You must now create the virtual channels on the ATM switch. In our example, you would enter these commands on SCP 1:

```
localhost::> conf vcc
localhost::configuration vcc> new 1a1 0 70 1a2 0 70
localhost::configuration vcc> new 1a2 0 70 1a1 0 70
```

1a1 is the switch port for node1 and **1a2** is the switch port for node2.

The switch completes the PVC based on the input port to output port virtual channel connection (VCC) mapping. Note that the PVC *vpi.vci* (**0.70**) matches the *vpi.vci* given to the atmarp commands on the hosts.

The above commands will connect two ports on the same ATM switch board. The Fore ASX-1000 switch has up to four switch boards. If you are connecting machines on different switch boards, the procedure is more complicated, as you must connect each port to the switch fabric and connect the fabric to each port. Thus, if you wish to connect a machine on port **1a1** to a machine on port **3a1**, the following commands are required:

On SCP 1:

```
localhost::> conf vcc
localhost::configuration vcc> new 1a1 0 70 1e3 0 70
localhost::configuration vcc> new 1e3 0 70 1a1 0 70
```

On SCP 3:

```
localhost::> conf vcc
localhost::>configuration vcc> new 3a1 0 70 3e1 0 70
localhost::>configuration vcc> new 3e1 0 70 3a1 0 70
```

On the Fore switch, the fabric connections are slot e. Therefore, port **1e3** refers to a connection from switch board 1 to switch board 3. Likewise, **3e1** refers to a connection from switch board 3 to switch board 1. Fore refers to these ports as "intra-fabric" ports.

Testing the Connections

Once the Classical IP setup is complete, all of the standard network tests can be performed. The simplest test is done by using the ping command to test the connection. One difference between SVC and PVC connections is a large latency for the first ping response when using SVCs. The reason for the latency is the setup time needed to establish the SVC. After the SVC is established, the latency for SVC and PVC connections should be the same.

After verifying the basic connectivity, you can run some network performance tests over the ATM connection. I have used the Netperf tool (see Resources) as well as some benchmarks developed locally. The maximum throughput performance is very good, around 132Mbps. This number is close to the maximum payload data rate for an OC-3 ATM network.

Conclusion

I have given instructions needed to set up the switch and hosts on an ATM network with Linux. The configuration steps given are specific to IP over ATM connections using the Classical IP standard. In addition to Classical IP, LAN Emulation (LANE) can be used to carry IP over ATM. LANE is supported by the Linux-ATM software as well, but configuration of LANE is beyond the scope of this article. For more information, refer to the documentation in the Linux-ATM distribution.

Hosts can communicate in several other ways using an ATM interface without relying on Classical IP. The ATM software supports "native" ATM sockets, where applications can communicate directly over an ATM connection, bypassing the IP software completely.

If you are interested in learning about ATM technology but don't have ATM hardware, the Linux-ATM software can be of help. The software has the

capability to emulate an ATM device using TCP/IP to make the actual connection. By taking advantage of this support, you can get a head start on configuring ATM for Linux and learning the ATM programming interface.

NIST

Resources

Wayne J. Salamon is a Computer Scientist in the High Performance Systems and Services Division at the National Institute of Standards and Technology in Gaithersburg, MD. He has worked on system software for PCs, UNIX workstations and IBM mainframes for the past 12 years. When not doing computer stuff, he appears to play guitar, though only when connected to vacuum tube amplifiers. Wayne can be reached at wsalamon@nist.gov.

LINUX JOURNAL

The GNOME Project

Miguel de Icaza

Issue #58, February 1999

What is GNOME and where is it heading? Miguel tells us all.

GNOME is an acronym for GNU's Network Object Model Environment. GNOME addresses a number of issues that have not previously been addressed in the UNIX world, such as:

- Providing a consistent user interface.
- Providing user-friendly tools and making them powerful by leveraging the UNIX foundation.
- Creating a UNIX standard for component programming and component reuse.
- Providing a consistent mechanism for printing.

GNOME's main objective is to provide a user-friendly suite of applications and an easy-to-use desktop. As with most GNU programs, GNOME has been designed to run on almost all strains of UNIX-like operating systems.

History of GNOME

The GNU GNOME project was initially announced in August 1997. After just one year of development, approximately two hundred programmers worldwide are now involved in the project.

The original announcement called for developers to shape the GNOME project in a number of forums: the GNU announce mailing lists; the Guile mailing list; and the GTK+ and GIMP mailing lists. The programmers and other people who influenced the project were mainly free software enthusiasts with diverse areas of expertise, including graphics programming and language design.

The GNOME team has been working steadily toward creating a foundation for future free software development. GNOME provides the toolkit and reusable component set to build the free software end users are eager for.

Our recent releases of the GNU Network Object Model Environment have been GNOME 0.20, the first version of GNOME that showed signs of integrations, released in May 1998; the Drooling Macaque 0.25 release, with more features; and finally, our latest public release, GNOME 0.30, codenamed Bouncing Bonobo.

The GNOME 0.20 release was the first release included in a CD-ROM distribution. Red Hat 5.1 shipped with a technology preview of the GNOME desktop environment and it was first demonstrated at the 1998 Linux Expo in North Carolina.

Before the Drooling Macaque release, GNOME software releases were coordinated by two or three people on the team. This became a significant burden, as precious time was being used coordinating each release. We have been trying to make the release process more modular and have assigned different modules to package maintainers. Each package maintainer is responsible for packing, testing and releasing their packages independently of the main distribution, which we consider to be the core libraries and the core desktop applications. So far we have had some success, but there is still room for improvement. We will continue to polish the release process to make it simpler.

Figure 1

Figure 2

The most recent GNOME release, Bouncing Bonobo, is the first to feature the GNOME spreadsheet Gnumeric.

Red Hat Advanced Development Labs

In January 1998, Red Hat announced the creation of the Red Hat Advanced Development Laboratories (RHAD). The initial objective of Red Hat Labs was to help the GNOME effort by providing code and programmers and by helping us manage the project resources.

All code contributed to GNOME by Red Hat Advanced Laboratories has been provided under the terms of the GNU GPL and the GNU LGPL licenses. Several GTK+ and GNOME developers have been hired by Red Hat and they have rapidly provided the GNOME project with a number of important features.

For example, Rasterman has implemented themes for GTK+; the GTK+ themes allow a user to change the appearance of the widgets. This is done by abstracting the widget drawing routines from the toolkit and putting those drawing routines in modules that can be loaded at runtime. Thus, the user can change the appearance of applications without shutting them down or restarting the desktop.

Figure 3

GTK+ themes are fully working. So far, a number of theme front-ends have been written. At the time of this writing, available themes include Motif, Windows95, Metal, native-GTK+ and a general purpose Bitmap-based engine (see Resources). The web site http://gtk.themes.org/ keeps an up-to-date list with many contributed themes from which to choose.

Various important changes to the GTK+ toolkit required for the GNOME project, such as the menu keyboard navigation code and the enhanced “Drag and Drop” protocols (XDND and Motif DND), were written by Owen Taylor, a famous GTK+ hacker now working for Red Hat Labs.

Assorted applications were created or are maintained nowadays by the GNOME team at RHAD as well: the Ghostscript front end (by Jonathan Blandford), the GNOME Help Browser and the GNOME RPM interface (Marc Ewing and Michael Fullbright), the GNOME Calendar and GNOME Canvas (Federico Mena) and the ORBit CORBA 2.2 implementation (Elliot Lee).

Other Donations

The GNOME project received a monetary donation from the GNU/Linux Debian team in the early stages of the project, as well as an Alpha board from Quant-X Service and Consulting GmbH. We are very grateful for their contributions.

Some Key GNOME Features

The GNOME libraries provide a framework to create consistent applications and to simplify the programmer's task. More features of the GNOME libraries will be described later. Some of the most important current developments in the GNOME libraries are discussed here.

Metadata

One problem faced in a desktop environment is the fact that it is usually necessary to have a mechanism for storing information about a file's properties. For example, applications might want to bind an icon for a specific

executable file, or bind a small thumbnail image for a graphic produced by a graphics program. These icons should be semantically attached to the main file.

The Macintosh OS, for example, provides a way to store this information in the file as its “resource fork”. This mechanism would be awkward at best to implement in a UNIX environment. The main problem is that a non-metadata-aware application can cause the metadata information to get out of sync.

The GNOME metadata was implemented by Tom Tromey at Cygnus, given a number of design constraints and tradeoffs (described in detail on their web site). The following is a list of the GNOME metadata features:

1. Binding the information on a per-file basis in a per-user setting, and each user keeps track of his own bindings. System defaults apply on top of these.
2. Binding information by file content is done according to the file type using file signatures, similar to the UNIX **file** command.
3. Binding information by a regular expression: for example, a default icon for gif files would be provided by the regular expression **.*\.gif$**.
4. The metadata system is optimized to provide a coherent GUI solution, rather than as a compromise or kludge to existing command-line tools.
5. Most ordinary uses of files will continue to work without metadata, just as they do now.

A number of standard properties for file metadata are available in GNOME. For example, “View” stores the action for viewing the file contents; “Open” stores analogous action for editing; “Icon”, which contains the icon, is used for displaying the file on the desktop.

Metadata types are MIME types.

Canvas

GNOME provides a **Canvas** widget, patterned after Tk's excellent canvas. This widget simplifies the programming of applications that need control over graphical components. The most noticeable feature of GNOME Canvas is that it provides a flicker-free drawing area where high-level objects can be inserted and manipulated. Basic zoom and scroll facilities are also a part of the canvas.

Figure 4

The high-level objects inserted into the canvas behave like regular widgets. They can receive X events, grab the focus and grab the mouse just like a regular

widget. As with their Tk counterparts, GNOME Canvas items can have their properties changed at runtime with a Tk-like configuration mechanism.

The GNOME Canvas ships with a number of items derived from the **GnomeCanvasItem** object: lines, rectangles, ellipses, arrows, polylines and a generic widget container to embed GTK+ widgets within a canvas. The Canvas framework is designed to be very extensible. As proof of this extensibility, the GNOME spreadsheet is implemented on top of the base canvas engine, with additional functionality provided by spreadsheet-specific **CanvasItems**.

Note that the current Canvas uses Gdk primitives (a thin wrapper over Xlib primitives) to draw, so it is limited in the quality and range of special effects that can be provided with it, which bring us to the next step in Canvas technology.

Raph Levien is working on an advanced rendering engine for the Canvas. It was originally developed as a stand-alone widget within his Type1 outline font editor, **gfonted**. At the time of this writing, work on integrating the engine into the Canvas was getting underway.

Features of this engine include:

- Anti-aliased rendering of all items
- Alpha transparency
- Items for vector and bezier paths
- Items for RGB and RGB plus alpha images
- Vector operations, including clip (intersect), union, difference and stroke layout
- PostScript Type1 font loading and rendering

The engine's design goal is to support almost all of the PostScript imaging model with the addition of alpha transparency. As such, it is expected to be an excellent starting point for high-powered graphics applications.

In spite of the ambitious goal of keeping the display up to date with entirely anti-aliased and alpha-composited items, performance is surprisingly good—comparable, in fact, to the Xlib-primitive-based canvas engine.

His code is expected to be merged into the main Canvas sometime soon.

Window Manager Independence

GNOME does not depend on a special window manager—any existing window manager will do. GNOME specifies window manager hints that can be

implemented by the window manager to give the user better desktop integration, but they are optional. The E window manager implements all of the GNOME window manager hints and can be used as a reference implementation for people wishing to extend their window managers to be GNOME-compliant. The ICEWM manager is tracking those developments and is also considered to be a GNOME-compliant window manager, although at this time, it is lagging a bit behind. A few people have shown interest in providing the WindowMaker and FVWM2 maintainers with patches to make those window managers GNOME-aware.

Component Programming

Historically, one of the attractions of UNIX has been the philosophy of small tools that each do one thing well and combining these tools, using pipes and simple shell scripts, to perform more complex tasks. This philosophy works very well when data objects are represented as plaintext and operations are effectively filters. However, this UNIX command-line philosophy does not scale well to today's world of multimedia objects.

Thus, it would be nice to have a framework in GNOME that would provide software reuse and component plugging and interaction, i.e., connecting small specialized tools to carry out complex tasks. With this infrastructure in place, GNOME applications can once again return to the UNIX roots of simple, well-specialized tools.

An RPC system was then required for providing this sort of functionality, so we decided to use CORBA (the Common Object Request Broker Architecture) from the Object Management Group (OMG). CORBA can be thought of as an object-oriented RPC system, which happens to have standardized bindings for different languages.

CORBA opened a range of applications for us. Component programming allowed us to package programs and shared libraries as program servers that each implement a specific interface.

For example, the GNOME mail program, Balsa, implements the **GNOME::MailMessage** interface that enables any CORBA-aware program to remotely compose and customize the contents of a mail message and send it. Thus, it is possible to replace the mail program with any program that implements the **GNOME::MailMessage** interface. As far as the GNOME desktop is concerned, the process just implements the **GNOME::MailMessage** interface. This means, for example, that I will be able to continue using GNUS to read my mail and have GNUS completely integrated with the rest of my desktop. This also applies to the other components in the GNOME system: the address book,

file manager, terminal emulation program, help browser, office applications and more.

Beside providing the basic GNOME interfaces, applications can provide an interface to their implementation-specific features. This is done by using CORBA's interface inheritance. A specific interface would be derived from the more general interface. For example, GNUS would implement the **GNOME::MailMessage** interface and extend it with GNUS-specific features in the **GNOME::GnusMailMessage** interface. This interface would hypothetically allow the user to customize GNUS at the Lisp level, something other mailers may not do. Another example would be a **GNOME::MozillaMailMessage** interface that would let a user configure the HTML rendering engine in Mozilla mail.

Not only does CORBA address these issues, but it can also be used as a general interprocess communication engine. Instead of inventing a new ad-hoc interprocess communication system each time two programs need to communicate, a CORBA interface can be used.

Embedding documents into other documents has been popularized by Microsoft with their Object Linking and Embedding architecture. A document-embedding model similar in spirit is being designed for GNOME (the Baboon model), and all of the interprocess communication in this model is defined in terms of CORBA interfaces.

Initially, we were very excited by the possibilities CORBA presented us, but we soon realized that using CORBA in the GNOME desktop was going to be more difficult than we expected.

We tried using Xerox's ILU for our CORBA needs. The license at the time did not permit us to make modifications to the code and redistribute them, an important thing for the free software community, so we had to look for alternatives. Xerox has since changed the licensing policy.

After evaluating various free CORBA implementations, we settled on MICO, as it was the most feature-full free implementation. MICO was designed as a teaching tool for CORBA, with a primary focus on code clarity.

Unfortunately, we soon found that MICO was not a production-quality tool suitable for the needs of GNOME. For one, we found that the rather indiscriminate use of C++ templates (both in MICO and in MICO-generated stubs) proved to be a resource hog. Compiling bits of GNOME required as much as 48MB of RAM for even the simplest uses of CORBA, and this was slowing down our development. Another problem was that MICO supported only the

C++ CORBA bindings. Even though an initial attempt had been made at providing C bindings, they were incomplete and not well-maintained.

To address these problems, Dick Porter at i2it and Elliot Lee at Red Hat Labs wrote a C-based, thin and fast CORBA 2.2 implementation called ORBit. As soon as ORBit became stable, the use of CORBA throughout GNOME began, after a delay of almost eight months.

With an efficient, production quality CORBA implementation under our control, we can ensure that CORBA-enabled interprocess communication is a valuable service for application programmers, rather than a source of overhead and bulk.

Dissecting a GNOME Desktop Application

The toolkit

GNOME desktop applications have been built on top of the object-oriented GTK+ toolkit originally designed as a GUI toolkit for the GNU Image Manipulation Program (GIMP).

GTK+ has been implemented on top of a simple window and drawing API called Gdk (GTK Drawing Kit). The initial version of Gdk was a fairly thin wrapper around the Xlib libraries, but a port to Win32 and a port to the Y windowing system are presently in alpha stages.

GTK+ implements an object system entirely in C. This object system is quite rich in functionality, including classical single inheritance, dynamic creation of new methods and classes, and a “signal” mechanism for dynamically attaching handlers to the various events that occur in the user interface. One of GTK's great strengths is the availability of a wide range of language bindings, including C++, Objective-C, Perl, Python, Scheme and Tom. These language bindings provide access both to GTK+ objects and to new objects programmed in the language of choice.

An additional feature of GNOME is Rasterman's Imlib library. This library is implemented alongside Gdk and provides a fast yet flexible interface for loading and saving images and rendering them on the screen. Applications using Imlib have quick and direct access to PNG, GIF, TIFF, JPEG and XPM files as well as other formats available through external conversion filters.

The Support Libraries

C-based GNOME applications use the GLIB utility library. glib provides the C programmer with a set of useful data structures: linked lists, doubly linked lists,

hash tables (one-to-one maps), trees, string manipulation, memory-chunk reuse, debugging macros, assertion and logging facilities. glib also includes a portable interface for a dynamic module facility.

The GNOME libraries

The GNOME libraries add the missing pieces to the toolkit to create full applications, dictate some policy, and help in the process of providing consistent user interfaces as well as localizing the GNOME applications so they can be used in various countries.

The current GNOME libraries are GTK+-Xmhtml, gnome-print, libgnome, libgnomeui, libgnorba, libgtop, gnome-dom and gnome-xml. Other libraries are used for specific applications: libPropList (soon to be replaced by a new configuration engine) and audiofile.

The main non-graphical library is called libgnome. This provides functions to keep track of recently used documents, configuration information, metadata handling (see below), game score functions and command-line argument handling. This library does not depend on the use of a windowing system.

As we use CORBA to achieve parts of our desktop integration, we have a special library called the libgnorba library to deal with various CORBA issues. It provides GUI/CORBA integration (to let our GUI applications act as servers), authentication within the GNOME framework and service activation.

The gnomeui library, on the other hand, has the code that requires a window system to run. It contains the following components:

- The GNOME session management support
- Widgets, both as straightforward extensions of GTK+ and designed to be dependent on libgnome features
- A set of standard dialog boxes otherwise not available on GTK+, well-integrated with other GNOME libraries
- Standard property configuration dialog boxes
- Standard top-level window handling
- A multi-document interface (gnome-mdi)
- Windowing hints
- CORBA integration where required

GTK+-XmHTML is a port of the Koen D'Hondt's XmHTML widget for Motif and is used for our HTML display needs. Our changes are being folded back into the main distribution.

The lib gtop library allows system applications to be easily ported to various operating systems; it provides system, process and file system information.

gnome-xml provides XML file loading, parsing and saving for GNOME applications and is being used in the GNOME spreadsheet (**Gnumeric**) and in the GNOME word processor program. **gnome-dom** provides an implementation of the World Wide Web Consortium's Document Object Model for GNOME applications. By the time you read this article, gnome-dom will have been deployed widely in the GNOME office applications. Both gnome-xml and gnome-dom were developed by Daniel Veillard from the World Wide Web Consortium.

Figure 5

gnome-print implements GNOME's printing architecture. It consists of a pluggable rendering engine as well as a set of widgets and standard dialog boxes for selecting and configuring printers. In addition, gnome-print is responsible for managing outline fonts and contains scripts that automatically find fonts already installed on the system.

The GNOME print imaging model is modeled after PostScript. Basic operations include vector and bezier path construction, stroking, filling, clipping, text (using Type1 fonts, with TrueType to follow shortly) and images.

Currently, gnome-print generates only PostScript output. However, the design of the imaging model is closely synchronized with the anti-aliased rendering engine for the Canvas and it is expected that these two modules will be interoperating soon. In particular, it will be possible to “print” into a canvas (useful for providing a high-quality screen preview) and to print the contents of a canvas. This feature should simplify the design of applications which use the Canvas, as very little extra code will be needed to support printing.

The same rendering engine will be used to render printed pages directly without going through a PostScript step. This path is especially exciting for providing high-quality, high-performance printing to color ink-jet printers, even of complex pages containing transparency, gradients and other elements considered “tricky” in the traditional PostScript imaging model.

Bindings

One explicit goal of GNOME was to support development in a wide range of languages, because no single language is ideal for every application. To this end, bindings for both GTK+ and the GNOME libraries exist for many popular

programming languages, currently C, C++, Objective-C, Perl, Python, Scheme and Tom.

The early involvement of Scheme, Tom and Perl hackers in both the GTK+ and GNOME projects has helped in making the GTK+ and GNOME APIs easy to wrap up for various languages. Multi-language support is “baked in” to the design of GTK+ and GNOME, rather than being added on as an afterthought.

Development model

GNOME is developed by a loosely coupled team of programmers around the world. Project coordination is done on the various GNOME mailing lists.

The GNOME source code is kept on the GNOME CVS server (cvs:cvs.gnome.org:/cvs/gnome/). Access to the source code through Netscape's Bonsai and LXR tools is provided at http://cvs.gnome.org/, to help programmers get acquainted with the GNOME source code base.

Most developers who have contributed code, major bug fixes and documentation to GNOME have CVS write access, fostering a very open atmosphere. GNOME developers come from a wide range of backgrounds and have diverse levels of skill and experience. Contributions from less experienced people have been surprisingly helpful, while the older, wiser coders have been happy to mentor younger contributors on the team. The GNOME developer community values clean, maintainable code. Even programmers with many years of coding experience have noted how involvement with the GNOME project has helped them write better code.

The GNOME Office Suite Applications

As the GNOME foundation libraries become more stable, the development of larger programming projects has become possible and has allowed small teams of developers to put together the applications which will make up the GNOME office suite.

As with other GNOME components, the GNOME office suite is currently catching up with commercial offerings. By providing an office suite which is solid, fast and component-based, the code written for the GNOME project might become the foundation for a new era of free software program development.

The office suite leverages a lot of knowledge many of us have acquired during the past year while developing various GNOME components. Our coding standards are higher, the quality is better and the code is more clean and more robust.

The availability of these applications has provided us with the test bed we required to complete our document embedding interfaces (the Baboon model).

Two word processing projects are going on for GNOME: one of them is GWP by Seth Alves at the Hungry Programmers and the other one is Go from Chris Lahey. GWP is currently more advanced and has printing working with the GNOME printing architecture.

Gnumeric, the GNOME spreadsheet project, is aimed at providing a commercial-quality spreadsheet with advanced features. It provides a comfortable and powerful user interface. As with other components in GNOME, we have worked toward providing a solid and extensible framework for future development.

Recently, work has begun on Achtung, the GNOME presentations program. It is still in the early stages of development.

Getting GNOME

Tested source code releases of GNOME are available from GNOME's ftp site: ftp://ftp.gnome.org/.

It is also possible to get the very latest GNOME developments from the anonymous CVS servers. Check the GNOME web page for details on how to pull the latest version straight from the CVS servers.

Breaking news about GNOME is posted to the GNOME web site in http://www.gnome.org/, along with documents to get you started on GNOME and developing GNOME applications.

Resources

Acknowledgments

Miguel de Icaza is one of the GNU Midnight Commander authors as well as a developer of GNOME. He also worked on the Linux/SPARC kernel port. He can be reached via e-mail at miguel@gnu.ai.mit.edu.

Archive Index Issue Table of Contents

Advanced search

KDE: The Highway Ahead

Kalle Dalheimer

Issue #58, February 1999

Mr. Dalheimer describes some of the plans being made for future versions of KDE.

The K Desktop Environment (KDE, see http://www.kde.org/) has already generated a lot of interest, and many individual users and institutions alike are using it as their desktop environment of choice. However, nothing is so good that it doesn't have room for improvement. Since the beginning, we have considered stability more important than sticking to announced dates, so there have been occasional delays when we considered KDE not stable enough for a release. That said, we hope to release KDE 1.1 after one or two beta versions by the end of 1998. This version will contain a number of bug fixes, the much sought-after support for ICQJava, more robustness in the HTML display code and a few new features such as configurable key bindings for the whole desktop. Originally, we had planned to release only bug fixes as version 1.0.1. However, the completed new features have been requested many times and we don't expect to have 2.0 stable for some time, so we want to include everything that is ready. This way, we can move on at full steam after 1.1 is released.

Figure 1. KDE 1.1 Desktop

Figure 2. KDE 1.0 Desktop

KDE 2.0, the next “official” version, will probably be out in late summer 1999, but this is still uncertain. In the meantime, we will also release a first alpha version of KOffice (discussed below).

Planned Features for KDE 2.0

We plan substantial rewrites and reorganizations of KDE for version 2.0. This will probably lead to snapshots that are unstable or don't compile, but as always, KDE follows the open-source model closely and makes daily snapshots available via the FTP site and the CVSup server. (CVSup is a software package for distributing and updating source trees from a master repository on a remote server host.)

Some of the new code that will go into KDE 2.0 belongs to the area file manager/web browser/networking code. Most of the new code is already written and just waiting to be committed to the CVS after the release of KDE 1.1. Changes include a complete rewrite of the file manager and the networking code which provides a great improvement over the current version, the ability to browse compressed archives and a complete overhaul of the user interface of the file manager. Related to this is the HTML widget currently in the process of being reorganized to make it more adaptable to new HTML standards and to provide better support for JavaScript and a free Java virtual machine. (**kaffe**, http://www.kaffe.org/, is a candidate if it supports standard AWT, Abstract Window Toolkit, programs by then.) Most of this is not just vaporware or simply plans for the future, but existing code, part of which is also available via the CVSup server with anonymous access.

Another program to be completely rewritten is the mail program KMail. The new version, dubbed KMail 2, will be more robust and flexible with respect to various attachments and feature IMAP support.

Of course, there is not only revolution such as complete rewrites, but also evolution as seen in the continuous development of existing features. More and more configuration modules are being added to the control center and we plan to provide more modules geared specifically toward system administrators. We hope to be able to support both LinuxConf and COAS (Caldera Open Administration System), but are definitely in need of more skilled developers to help in this area.

Another area of improvement is the drag-and-drop protocol where we will switch from our “homegrown” protocol to the XDND protocol, since it will likely become the future drag-and-drop standard on (at least free) UNIX systems and is currently supported by programs written with the JX toolkit. The GNOME developers have indicated that they will also support this protocol, possibly leading to a first point of interoperability between the KDE and GNOME programs.

We will also make use of some new features provided by version 2.0 of Qt, the toolkit used with KDE. Among these are themes and Unicode support. Themes

are something I personally consider less important for a desktop aimed at improving productivity, but I know a lot of users want it and we aim to please. Note that some theme ability is currently in KDE (see e.g., http://kde.themes.org/), but the new code will extend this to single widgets within the application windows.

Plans have been made to tear the window manager engine out of the window manager KWM and put it into the library, so that there can be different window managers to implement a different look-and-feel and still provide the window manager functionality needed by a full-featured KDE desktop.

Unicode (see http://www.unicode.org/) is very important to gain acceptance in the Far East and other areas of the world with scripts containing more than 256 characters. Two Chinese localizations of KDE already exist, but these require a patched X server that combines two character codes transmitted for display into one. With Unicode support, such tricks will not be necessary, as the KDE message files can then contain 16-bit characters directly.

The usability of Unicode support is based heavily on availability of decent fonts for the script in question, something UNIX systems have traditionally been lacking. I have been looking at integrating the free True Type engine Freetype into KDE, but this is still in the beginning stages. (Contributions are welcome.) Another option is using a font server that supports True Type fonts.

Another area where work is being done is making KDE accessible for handicapped users. It is already possible to use very large fonts with KDE programs and especially to set such a large font once and for all for all KDE applications, so that people with slight vision impairment can use KDE programs, but this is not enough. We have had some success with voice-type software, i.e., software that allows users to navigate and operate the desktop and applications by speaking commands into a microphone. We are working with universities on leveraging their research results and making usable programs out of them. Another feature that falls into this category but has not been addressed yet is screen readers, i.e., software that reads out whatever is visible on your screen via your sound card and the speakers. While the necessary text-to-speech synthesis is a non-trivial problem (even though programs such as **emacspeak** show that it can be done in free software), screen readers become even more difficult to write when graphical user interfaces are involved, because a single text flow is no longer available as on terminal screens.

The topic most interesting to readers is the development of KOffice. KOffice (see http://koffice.kde.org/) is intended to be a complete office productivity suite like Microsoft Office or Lotus SmartSuite. It is CORBA-based (using the high-quality, freely available ORB Mico that makes good use of today's compiler technology) and uses the Open Parts as its object model. Open Parts is the master's thesis topic of one of our developers and provides a rich object model on top of the rather bare-bones CORBA standard, including addition of event services and event filtering to CORBA. Open Parts can even be used with non-KDE applications.

Not only the large KOffice applications will use CORBA, but this technology will also be used in other areas, including the panel which would then be better able to swallow other programs. On the other hand, adding CORBA compliance to a relatively small program like the panel means bloating it, and we still have to investigate whether this is truly worth it.

Figure 3. KPresenter

Figure 4. KSpread

Currently, KOffice consists of the following applications which, as I write, have different levels of usefulness. First, there is KPresenter (see Figure 3), a presentation program in the spirit of MS PowerPoint that has already been used for “real work” (like my KDE presentation at the last International Linux Congress in Cologne) and features a large number of blending effects. Then, there is KSpread (see Figure 4), a spreadsheet that already contains about 20% of the functionality of MS Excel and is easily extensible via scripts written in Python.

A recent addition to KOffice is KWord, a word processor aimed at smaller size documents like personal letters, as well as complete books. Besides normal word processing features, it also has some features known from DTP programs,

such as Adobe FrameMaker. For large documents or those containing many formulas, it might be better still to use KLyX, a WYSIAWYG (what-you-see-is-almost-what-you-get) document processor that uses LaTeX as its formatting engine and is based on its older brother, LyX. KLyX is not yet as fully integrated into the KOffice concept as the other applications, but hopefully it will be in the near future.

Figure 5. KIllustrator

Figure 6. KFormula and KImage

The KOffice application that is probably most advanced at the moment is KIllustrator (see Figure 5), a vector-based drawing package. It features a lot of drawing tools and is very usable. Two other applications of KOffice are KFormula (see Figure 6), a mathematical formula typesetter; and KImage, a smaller image-manipulation program. Finally, there is KChart, a component for drawing business graphs such as bar charts and pie charts; and KOrganizer, an organizer program in the spirit of Lotus Organizer that is quite usable and sports many features and different views.

Since all KOffice programs are based on the Open Parts framework, they can all be embedded into each other. You can nicely embed a spreadsheet document into your word processor document and beef the whole thing up with a vector drawing done with KIllustrator. This embedding also features in-place activation with dynamic menu bars, and unlike other embedding technologies on some platforms we like to hate, it is quite stable.

We are very aware that for Linux to become a viable alternative to Windows or the Macintosh, it must feature a consistent easy-to-use desktop and also have an office productivity suite that need not (or should not) be as bloated as Microsoft Office, but definitely needs the features used on a daily basis and must be able to read common document formats. Unfortunately, Microsoft Office formats are not well-documented. Still, with the help of the work done in LAOLA (library for Microsoft structured storage files), we hope to be able to support at least some of them. Even though I think this is a boring task (if somebody thinks I am wrong here, you are welcome to help with this project), we will also write an RTF reader and writer, and I have already successfully imported some FrameMaker documents via my homegrown filter into KWord. Whether other formats (such as WordPerfect, StarOffice or the Lotus SmartSuite) will be supported depends on two things: whether the respective manufacturers will make their format documentation available and whether someone volunteers to do the work.

Summary

As you can see, many things are happening in the KDE world and development is occurring faster than ever. As with all free software projects, we can be only as good as the people who support us, so if you are interested in GUI hacking, writing file-format filters or any of the other topics I have mentioned (or something completely different that fits into the overall KDE concept), don't hesitate to contact us (see http://www.kde.org/ for contact information). This is truly an exciting project to be in.

Kalle Dalheimer is a freelance software consultant and a technical writer/ technical editor for O'Reilly. He is a member of the KDE core team, where he is in charge of the libraries and some of the applications. When not hacking or writing, he plays with his two-year-old son, setting up wooden miniature railway systems. He can be reached via e-mail at kalle@dalheimer.de.

Archive Index Issue Table of Contents

Advanced search

P-Synch

Tim Parker

Issue #58, February 1999

P-Synch uses a number of scripts and API routines to perform all the password changes on your network from a single location.

- Manufacturer: Mercury Information Technology, Inc.
- E-mail: sales@m-tech.ab.ca
- URL: http://www.m-tech.ab.ca/
- Price: Customized
- Reviewer: Tim Parker

One of the annoyances of administering networks is the need to change passwords regularly. We change passwords often for security reasons and that's fine. What bothers me is the need to log in to each machine individually and run the password changing programs on each one, then log into each individual application that has its own user and password lists independent of the host machine and change the passwords there. My office network requires over thirty such changes; my home network, eight.

If I were running only UNIX, then I could use NIS and let that service change my machine passwords for me. However, NIS doesn't change non-UNIX passwords and doesn't do anything for application passwords. The same applies to Windows NT-based domains, where a central user list is maintained. Domain users don't extend to UNIX or applications. Utilities exist to provide some cross-platform support for NIS and NT domains, but I haven't found one that works well across my mixed network platforms.

Obviously, this is a problem that has plagued users for years. The folks at M-Tech (Calgary, Alberta) have done something about it. M-Tech's solution is called P-Synch (for password synchronization). It uses a number of scripts and

API routines to perform all the password changes on your network from a single location. (It is an obvious solution, once you've seen it work.) I first encountered P-Synch two years ago in an earlier version and have used it religiously on my own networks, recommended it to many clients and written about it extensively since. A Linux implementation of P-Synch was an obvious spinoff of the UNIX version. Even better, M-Tech has released a new version of P-Synch which adds several new features that make it more functional and easier to manage.

As you may have gathered from the comments above, P-Synch uses scripts to change passwords on all applications and machines you tell it to access. P-Synch accomplishes this by using either a native API or TELNET to log in to each machine or application one at a time, running whatever commands are necessary to modify the passwords. You specify only the password to change to (or to reset) and P-Synch runs through its list of targets in the background. Since only a single copy of P-Synch is required anywhere on the network, no client programs need to be installed on each machine. The user interface can be character-based, GUI or HTML. An administrator defines which machines and applications a user can modify passwords on, as well as more advanced options such as password aging.

Since P-Synch is script-based, it can change passwords on any machine or application that can be logged in to using TELNET or for which an API can be written. The list of M-Tech-provided scripts for operating systems is lengthy: Linux, many versions of UNIX, NetWare, Windows 95/98 and NT, LAN Manager, PathWorks, MVS and VMS. Application scripts available from M-Tech include Oracle, Sybase, SQL-Server, cc:Mail, MS-Mail, Exchange, GroupWise and GroupWare (including Lotus Notes). Scripts for other targets are starting to appear in Usenet newsgroups and on web pages, mostly written by P-Synch administrators.

The documentation for P-Synch may be downloaded from the M-Tech web site in PDF, PS or HTML formats. You can download uncompressed, ZIP or GZIP versions. The installation and configuration guide is about 260 pages long, but you will likely need to examine only a few pages to install P-Synch properly. Instead of printing the entire document, an Acrobat or PostScript viewer is best used to find those sections of interest. (Viewing on-line saves both time and paper.)

If you are running NIS (or a Windows-based alternative), P-Synch needs to run on the master. P-Synch will not function properly if installed on a client of an NIS master. (NIS does not allow administrative password changes from clients.) If you are not running NIS or a similar network-wide user management system, P-Synch can reside on any machine on the network. All machines must be

running TCP/IP. P-Synch can coexist quite happily with NIS, handling the non-NIS targets only, if you prefer.

Prior to installing P-Synch, you must gather a list of the IP addresses of each machine on which P-Synch will change passwords. P-Synch requires root (or equivalent) access on each of these machines. It is handy to create a test account or two on the server and a few clients to make sure P-Synch is performing its tasks properly before trusting it to your network-wide password management. These test accounts can be deleted after testing or saved for other purposes. A complete installation takes about 4MB of disk space (noticeably down from the last version's requirement of 9MB).

Installing P-Synch takes only a few minutes; however, configuring for a network with several applications and operating systems can take a while longer. The usual installation procedure is to copy or download the files to your Linux machine and extract the files in the library (tar, gzip or zip). One such library file is called unix_srv.tar and contains binaries for all supported UNIX and Linux platforms. After extracting this tar file, an installation shell script is run which handles the file setup procedure. A manual check of a configuration file to ensure it has the proper location for the passwd file (by default, it assumes /etc/passwd) completes the file setup. P-Synch normally uses TCP port 106, but this can be changed if port 106 is in use by another service. To test the installation, a TELNET session to the TCP port should produce a password prompt, at which point the installation is finished. The installation guide contains a list of common problems encountered when setting up P-Synch and they should account for most Linux system configurations.

P-Synch uses a script called psynch.conf to manage password changes. Separate parts of the program control users changing their own passwords, as well as root changing any password. On Linux and UNIX systems, P-Synch modifies passwords directly by interaction with the passwd program, not through a script (which would provide a potential security hole). The psynch.conf script can be edited if necessary, which makes it easy to handle special requirements such as firewalls and proxies as well as encryption schemes that manage password files. For non-UNIX passwords, P-Synch's psynch.conf file must be modified to use a script for password changes. M-Tech provides a number of prewritten scripts for different operating systems as well as applications that reside on UNIX or Windows machines (such as Oracle, Lotus Notes and so on). Non-UNIX systems require changes to the inetd file, but these can be cut-and-pasted from M-Tech's documentation or scripts. Linux and UNIX versions of P-Synch require a login called psadmin, which is used by the server to verify that P-Synch agents on other machines are allowed to change passwords. The psadmin login should be set up so that it has no access privileges.

Our test network consisted of three Linux machines (two Red Hat and one Slackware), four that were SCO OpenServer or UnixWare and three Windows NT servers, along with twelve Windows 95/98 machines. The server applications were Oracle 8, Lotus Notes, Exchange, Novell GroupWise and SQL-Server (all on the Windows NT servers). On this network, installation and configuration of P-Synch took about two hours. Most of that time was spent setting up the non-Linux/non-UNIX password change routines, with about half an hour required to debug the various scripts. If you are working on a Linux-only network, the process will take less than ten minutes.

Once properly configured, users anywhere on the network could run the P-Synch routines to modify their own passwords. The HTML interface in particular is friendly and attractive. Users can specify which machines or applications the change affects, or accept all (the usual case). Administrators can change passwords from any client. The amount of time required for a password change depends on the network load, the number of targets and the nature of the operating systems. On our test network, the password changes went quickly, completing in under two minutes.

To test P-Synch with NIS, we set up one of the SCO machines as an NIS master and a Red Hat Linux system as the slave. We let NIS handle the password changes on half of the UNIX and Linux machines while P-Synch handled the rest of those types as well as the Windows machines. Propagation time for password changes didn't drop noticeably, which was expected since the Windows and application scripts are the major time consumers.

P-Synch is usually licensed to networks based on the number of users. M-Tech will customize the pricing plan to suit your requirements. Earlier versions of P-Synch cost about $10 per user; the latest version is likely to be similarly priced. If you don't want to worry about the password-changing hassle anymore, you'll find P-Synch to be a wonderful utility. It is fast, clean, easy to use and worth every penny. The benefits are multiplied many-fold on heterogenous networks. If you want more information about P-Synch, or want to download the application itself for evaluation, check out the M-Tech Web site at http://www.m-tech.ab.ca/. For the curious, a white paper describing P-Synch is available in Acrobat PDF and PostScript formats.

Tim Parker lives in Ontario, Canada, and can be reached via e-mail at tparker@tpci.com. He is a widely published UNIX author with over 1,000 articles and 40 books on the subject. Dr. Parker's latest book is Linux Unleashed, Third Edition published by Sams. When not writing, Tim flies planes, scuba dives and argues with his temperamental network of thirty PCs and workstations. He often loses the arguments.

The login Process

Andy Vaught

Issue #58, February 1999

The beginning of it all....

Virtually all Linux sessions begin with the user typing his user name at a prompt that looks like this:

```
login:
```

In this article, I will explain a little about what really happens behind the scenes and what contortions the system goes through to get a user going.

A Bit about the Shell

First, a quick look at the shell. The shell, which is just a program like any other, reads the characters you type and looks for a program with the same name. Program names are typed at the prompt and executed by the shell. Ending a command line with the **&** character causes the command to be run in the background.

The shell runs a program in two steps. First, the shell does an operation called a “fork”. Forking creates a new process that looks just like the original process, inheriting many attributes of its parent such as any open files and user ID. Although it is an exact copy of the shell program, the “child” process does not read user commands. The child shell immediately does an operation called an “exec”, short for “execute”, in which it causes the Linux kernel to load the new program over the top of the child shell and run that program in its place.

At this point, the original shell simply waits for the child program to finish. Once done, it gets the next line of input from the user, then the whole procedure is repeated. In an active UNIX system, this sort of thing is happening all the time. Even on fairly inactive systems, processes are still run to do housekeeping chores, while others are simply sleeping and waiting for something to happen.

From the **bash** shell, you can see how **exec** works by typing

```
exec ls -l
```

The **ls** command runs as usual, but when it is done, you are no longer logged in. The shell is replaced by ls, and when it finishes, it is as if your shell had finished.

How Does it All Get Started?

When the kernel is first loaded into memory, it initializes itself and any hardware that may be attached to the computer. Once the kernel is established enough to be able to run programs, it does. The first program is called “init”; its job is to function as the ancestor of all processes.

When **init** starts, it reads a file called inittab, usually located in /etc. This file tells init which programs should be run under which conditions. Not only does init run the startup scripts that bring the rest of the system up, but init also takes care of shutting the system down.

One of the responsibilities of init is running the programs that let users log into the system. For a terminal (or virtual console), the two programs used are **getty** and **login**. **getty** is short for “get terminal”. A basic getty program opens the terminal device, initializes it, prints **login:** and waits for a user name to be entered. Modern getty programs (several are available for Linux) can do other things as well—if the terminal device is a (recent) modem, they can read status codes sent by the modem to tell if the call is voice or fax and handle the call appropriately. Most of the time, though, someone just wants to log in, so getty executes the login program, giving the user name to log in via the command line.

The **login** program then prompts the user for a password. If the password is wrong, login simply exits. The init process then notices that one of its children has exited and spawns another getty process on the terminal in question. If the password is good, login changes its process user ID to that of the user and executes the user's shell. At this point, the user can type commands. When the user exits by typing the shell's built-in **logout** command, the shell exits and init notices that its child has exited and spawns another getty on the terminal.

Why are two separate programs used to log in instead of just one? The answer is that doing it this way provides more flexibility. For example, getty doesn't have to execute login—it can execute a program to receive (or send) faxes, a PPP daemon to emulate a network connection over a serial line, or if you have a modem with “voicemail”, one of those phone tree programs that people hate so much (“press five to hear these options again”).

Similarly, login is sometimes needed without getty; for example, when a user logs in over a network, no terminal device is waiting. Instead, each new connection is handled by a program called **telnetd** that forks and executes a login process. **telnetd** remains to pass characters between the network and the new shell.

As a partial example of how the process works, Listing 1 shows an **autologin** replacement for getty. This replacement is meant for people who are tired of typing their user ID and password for the bazillionth time. You can boot Linux and have it drop straight into a couple of shells—sort of like DOS, but with virtual consoles.

To install autologin, copy it to the /sbin (system binaries) directory and type:

```
chmod +x /sbin/autologin
```

as root. Still as root, edit the /etc/inittab file and change the lines that look like this:

```
c1:12345:respawn:/sbin/getty 38400 tty1
```

to:

```
c1:12345:respawn:/sbin/autologin tty1 login -f myid
```

replacing **myid** with your own user ID. Red Hat installations typically do not have the letter c at the beginning of the line.

Be sure to leave some of the lines containing getty exactly as they are—if you do something wrong, you are going to need a way to log into your system. On my own system, I change **c1** through **c3** and run three initial shells. Once the file is edited, reboot the system and all should work.

The first argument to autologin is the name of the terminal. The rest of the command line is used as the login command that does the work.

A Synopsis of autologin

The first line tells the kernel how to run this program, in this case by letting the bash shell interpret it. The first exec line is a Bourne shell trick that lets a shell script change the source/destination of its standard input, standard output and standard error. We want to set file descriptors 0, 1 and 2 to refer to the terminal device as expected by login (and many other programs) when they run. The **cat** command displays the system's standard logon message. The **shift** command shifts the positional parameters to the shell script. Argument $1 is deleted, argument $2 becomes $1, argument $3 becomes $2 and so on. The

last line executes the rest of the command line as a program. In this case, the **login -f** option performs the normal login procedure, with the **-f** option telling login not to bother with passwords.

Andy Vaught is currently a Ph.D. candidate in computational physics at Arizona State University and has been running Linux since 1.1. He enjoys flying with the Civil Air Patrol as well as skiing. He can be reached at andy@maxwell.la.asu.edu.

Archive Index Issue Table of Contents

Advanced search

Caching the Web, Part 2

David Guerrero

Issue #58, February 1999

This month Mr. Guerrero tells us about the definitive proxy-cache server, Squid.

Last month we discussed the basic concepts of proxy servers and caching. Now, let's see how to implement this technology in your organization. A few proxy-server programs are on the market, such as MS-PROXY, aka Catapult, available only for Windows NT, and Netscape Proxy Server, available for different UNIX platforms and Windows NT. Both have two main drawbacks: they are commercial software and they don't support ICP. The excellent Apache web server has included a proxy-cache module since its 1.2 version. This module is a very interesting option: it's free, and works with the most popular web server on the Net. However, it doesn't use ICP, and its robustness is not comparable to the best choice for a proxy-cache server—**Squid**.

Squid is a high-performance proxy-cache server derived from the cache module of the Harvest Research Project, maintained by Duane Wessels. It supports FTP, gopher, WAIS and HTTP objects. It stores hot objects in RAM and maintains a robust database of objects in disk directories. Squid also supports the SSL protocol for proxying secure connections and has a complex access control mechanism. Another interesting feature of Squid is negative caching, which saves “connection refused” and “404 Not Found” replies for a short period of time (usually five minutes).

Squid consists of four programs:

- **squid**: the main proxy server
- **dnsserver**: a DNS lookup program that performs single, blocking DNS operations
- **unlinkd**: a program to delete files in the background from the cache directory

It also provides a CGI program, designed to be run through a web interface, that outputs statistics about its configuration and performance and allows some management capabilities.

Squid Installation

Installing Squid is easy. Just download the source archive from http://squid.nlanr.net/ and, in a temporal directory, type:

```
gzip -dc squid-x.y.z-src.tar.gz | tar xvf -
```

Next, compile and install the software by typing:

```
cd squid-x.y.z
./configure
make all
make install
```

These commands install all needed programs and configuration files to /usr/local/squid. The binary programs are installed in the /bin directory, the configuration files in /conf. Log files are located in the /logs directory, and the object database in the cache directory and its subdirectories. A shell script called **RunCache** is in the bin directory used to run the squid binary, and assures that if the process dies for any reason, it is restarted automatically. So, put the following line in your rc.local file:

```
/usr/local/squid/bin/RunCache &
```

This will generate an error log in /usr/local/squid/squid.out, if Squid could not start because of some configuration problem.

Of course you can choose to install an RPM version of Squid if you use RedHat Linux or another distribution that supports RPM packages.

Squid installs a sample configuration file called squid.conf with many comments for each option. Here you can change the ICP and HTTP ports (3128 by default) and define how much memory and disk space to reserve for caching objects and other parameters such as refresh patterns and access control restrictions. Of course, you need an ICP port only if your cache is going to be the sibling or parent of other caches. The directives for changing these values are **http_port**, **icp_port**, **cache_dir** and **cache_swap**. Additionally, you can set the maximum object size to be stored in the database; the default is 4MB. Also, you should uncomment the following lines in this file:

```
cache_effective_user nobody
cache_effective_group nobody
```

This avoids running Squid as root, a dangerous habit for anyone who runs servers like **httpd** or **gopherd**. If you are using a recent version of Squid (at the time of this writing, the current version is 1.1.16), it will not start running as root, but will write an error message to the squid.out file.

To let Squid use 100 MB of your HD, the directive **cache_dir** should be something like this:

```
cache_dir /usr/local/squid/cache 100 16 256
```

Before starting Squid for the first time, create the cache and logs directories. To build the cache and hashed subdirectories, you should execute the commands:

```
cd /usr/local/squid
mkdir cache
chown -R nobody cache
cd /usr/local/squid/bin
./squid -z
```

Finally, to create and change the owner of the logs directory:

```
cd /usr/local/squid
mkdir logs
chown nobody logs
```

Now Squid can be run safely for the first time, with the above RunCache invocation. It will spawn several **dnsserver** processes and write its PID in the file logs/squid.pid. Important warning or error messages can be found in the squid.out and logs/cache.log files. Remember, if you want to shut down the cache, you must first kill the RunCache process to avoid an immediate restart and then type:

```
/usr/local/squid/bin/squid -k shutdown
```

Never use **kill -9** to shut down the cache, because it doesn't close the object database in such a way that it can be recovered—you'll probably lose part of it.

Restricting Access to Your Cache

In order to enable only those users who are in your organization to access your cache, you must set up some access control lists (ACLs). Defining access lists in Squid is quite easy; all access lists are defined with a name and are used to define a subset of elements. You can make a subset of IP addresses, protocols, destination URLs and even browser brands. The directive to define an ACL or subset is:

```
acl
```

You can learn more about ACL types in the example squid.conf. In the case of restricting access to only our users, the type needed is **src**. For example, suppose you want to allow access to the cache to all browsers in the 172.16.236.0 class C, the first 32 addresses of the next class C and your PC, 172.16.237.180. You can define an ACL like this:

```
acl my_users src 172.16.236.0/255.255.255.0
acl my_users src\
172.16.237.1-172.16.237.32/255.255.255.255
acl my_users src\
172.16.237.180/255.255.255.255
```

Next, define an ACL for the rest of the addresses. This line is included in the squid.conf example file:

```
acl all src 0.0.0.0/0.0.0.0
```

Apply these ACLs in an ordered way with the **http_access** directive. The syntax is:

```
http_access
```

For example:

```
http_access allow my_users
http_access deny all
```

More than one ACL can be combined in the same http_access directive and can be used in its negative form (i.e., preceded by !). The example shown is the most simple use of ACLs, but more complex forms will allow connections only in designated hours and days, allow only defined URLs or domains to be fetched and restrict some protocols such as FTP. This powerful feature of Squid can help you enforce and implement your security policy, whether you use Squid in your firewall or the Squid machine is the only one allowed to cross your firewall. Just look for examples in squid.conf.

There is also an ACL to permit setting the desired web ports you allow your users to use. This is the **Safe_ports** ACL. You should uncomment this line and add the **443** port to this ACL in order to allow the use of secure web servers through your Squid server.

A Look at the Logs

Squid can generate huge logs of your proxy-cache usage. With this information and the help of some scripts, we can generate complete access statistics, like the ones generated from web servers. Squid maintains three main log files:

- **cache_log** includes warnings and information about the status and operational issues of the cache.

- **store_log** includes information about database operations, such as inserts of new items and releases of expired objects.
- **access_log** contains an entry for each object fetched from the cache and information on how it was served. It also includes information about each ICP query received by the cache from other servers using this server as a neighbor.

Many utilities are available for generating statistics from the access_log file (see Resources). Remember, it is not considered ethical to surf your access_log to see which places your users visit. Some sites have chosen not to publish processed statistics in any form to guard their users' privacy, which is an important concern for all of us involved in the Internet community.

The logs grow very quickly and in a few days can eat up your remaining disk space. To safely clean your log files, you should rotate them with the SIGUSR1 signal. A single line can be added to your crontab to begin new log files each night:

```
/usr/local/squid/bin/squid -k rotate
```

This command will create the files access_log.0, store_log.0 and cache_log.0 and begin logging to new empty files. Now you can safely remove these files or process them for statistical purposes. The next time you rotate logs, ***files.0*** will be moved to ***files.1*** and so on. You can configure how many extensions Squid will use for these rotations to save disk space with the **logfile_rotate *n*** directive in the squid.conf file.

Configuring Browsers to Use Cache

To begin using your new proxy-cache server, you must first instruct your user's browsers to fetch objects from your server instead of retrieving them directly. In most modern web browsers, one of the configuration options is the specification of the proxy setup. Another option is to specify a list of domains or URL patterns which must be fetched through the proxy.

In Netscape Navigator or Communicator, you can include a proxy server and its port for each service to be proxied. With Squid, you can use these settings for the HTTP, Security (SSL), FTP and WAIS services, all with the same port (3128, by default). First, select the “Manual proxy configuration” radio button and then the “View” button to type in your settings. Figures 1 and 2 show examples of these screens.

Figure 1. Proxy Preferences Screen

Figure 2. Manual Proxy Configuration Screen

Another solution is the Automatic Proxy Configuration, introduced in Netscape Navigator 3.0, that allows multiple proxy servers, backup servers and different servers by domains. This configuration sits in a Javascript-like file that must be retrieved from a server. Using it, you can change the topology of your cache mesh or introduce new servers that must be treated as “No proxy for” servers. Without telling your users to change their configurations, the new configuration script is reloaded each time the browser is launched. MS Internet Explorer has also supported the automatic proxy configuration feature since version 3.02.

Figure 3. Automatic Proxy Configuration Screen

An example of this kind of configuration for Netscape Navigator and Communicator is shown in Figure 3. In this example, each time the browser is started, it loads the file proxy.pac from the server intranet.mec.es. This file must be returned with MIME-Type application/x-ns-proxy-autoconfig which can be accomplished in two ways:

1. Or add the following line to your mime.types file:

```
application/x-ns-proxy-autoconfig pac
```

1. Add the following line to your Apache srm.conf file:

```
AddType application/x-ns-proxy-autoconfig pac
```

For the changes to take effect, you must name your proxy auto-configuration file with the .pac extension and restart your web server. The Netscape documentation will tell you about the syntax of the .pac file (see Resources). Nevertheless, we'll look at a couple basic examples of how to write them.

No HTML tags should be embedded in the Javascript file, just the function **FindProxyForURL** with arguments *URL* and *host*. This function should return a single string containing **DIRECT** (get the object directly from the source), or **PROXY host:port** (get the object through this server and port). The string can contain more than one of these directives, separated by semicolons. For example:

```
function FindProxyForURL(
{
return "PROXY proxy1.mec.es:3128;
PROXY proxy2.mec.es:80; DIRECT ";
}
```

will instruct the browser to use the first proxy to fetch the object. If it can't contact the first (proxy1), then it will try the second (proxy2); in the case that both are down, it will fetch the object from the source. This gives a fault tolerance level to our cache system.

One interesting feature is using different proxies for different domains and including support for internal servers where we don't want to use the cache. For example:

```
function FindProxyForURL(
{
 if ( isPlainHostName(host) || dnsDomainIs(host,
        "intranet.mec.es"))
 return "DIRECT";
 else if (shExpMatch(host, "*.com"))
 return "PROXY proxy1.mec.es:3128";
 else
 return "PROXY proxy2.mec.es:80";
}
```

This function will directly fetch all objects whose URL is only a word with no dots or the Intranet server, all .COM objects from proxy1 and the rest from proxy2.

As a tip, the .pac file can be generated “on the fly” by a CGI script, giving different proxy configurations for different browsers, e.g., depending on the REMOTE_HOST environment variable provided by the CGI interface. In this way, load balancing between different networks can be achieved. Always remember that the MIME-type returned by the CGI must be **application/x-ns-proxy-autoconfig**.

Joining a Hierarchy

If your cache is to be part of a cache mesh or your proxy server is to be connected to another proxy that will be its parent, you must use the **cache_host** directive. You must include one line for each of your neighbors. The syntax for this line is:

```
cache_peer
```

where:

- *hostname* is the name of your neighbor.
- *type* is one of parent or sibling.
- *http_port* is the neighbor's port from which to fetch objects.
- *icp_port* is the port to which ICP queries are sent. Use a value of 0 if your neighbor does not run ICP, or 7 if your neighbor runs the UDP echo service. This can help Squid to detect if the host is alive.

You can specify the option **default** to use this host as a last resort in case you can't speak ICP with your parent cache. Another option is the **weight=*N*** to favor a specific parent or sibling in the neighbor selection algorithm. Larger values give higher weights.

If you have a stand-alone cache, you should not include any of these directives. If you have one parent that runs its HTTP port on 3128 and its ICP port on 3130, the line to include in the squid.conf file is:

```
cache_peer
```

With the **cache_peer_domain** directive, you can limit which neighbors are queried for specific domains. For example:

```
cache_peer_domain
cache_peer_domain
```

will query the first cache only for the .COM and .EDU domains, and the second for some of the European domains.

If you have only one parent cache, the overhead of the ICP protocol is unnecessary. Since you are going to fetch all objects (HITs and MISSes) from the parent, you can use the **no_query** option in the **cache_peer** directive to send HTTP queries to only that cache.

Also, there are some domains you will always want to fetch directly rather than from your neighbors. Your own domain is a good example. Fetching objects belonging to your local web servers from a faraway cache is not efficient. In this case, use the **always_direct** ***acl*** command. For example, in our organization we use:

```
acl intranet dstdomain mec.es
always_direct allow intranet
```

to avoid getting our own objects from the national cache server.

The Cache Manager

Squid includes a simple, web-based interface called **cachemgr.cgi** to monitor the cache performance and provide useful statistics, such as:

- The amount of memory being used and how it is distributed
- The number of file descriptors
- The contents of the distinct caches it maintains (objects, DNS lookups, etc.)
- Traffic statistics with each client and neighbors
- The “Utilization” page, where you can check the percentage of HIT your cache is registering (and thus bandwidth you are saving).

Be sure to copy the cachemgr.cgi program installed in your /usr/local/squid/bin (or wherever you chose) to your standard CGI directory, and point your browser

to http://your.cache.host/cgi-bin/cachemgr.cgi. There, you should type your cache host name, usually “localhost” or the name of your system, and the port your cache is running, usually 3128, and check all the options.

Conclusions and Tips

A proxy-cache server is a necessary service for almost any organization connected to the Internet. In this article, we have tried to show the whys and hows to implement this technology, and a brief tutorial on Squid, the most advanced and powerful tool for this purpose. Don't forget to read all the comments in the example configuration file. They are complete and useful and show a lot of features not mentioned in this article.

Perhaps in a few years, with the growth of PUSH technology and the use of dynamic content on the Web, caching won't be a solution to the bandwidth crisis. Today, it's the best we have.

One problem proxy caches don't solve is making certain your users configure their browsers to use the caches. Users can always choose to bypass your proxy server by not configuring their browsers. Some organizations have chosen to block port 80 in their routers except for the system running the proxy-cache server. It's a radical solution, but very effective.

Another thing you can do to improve the speed of your users' browsers is pre-fetching the most accessed web sites from your cache. Recursive web-fetching tools which support proxy connections can help do this task in non-peak hours, e.g., **url_get**, **webcopy**. Launching one of these retrieval tools with the standard output redirected to /dev/null updates the cache with fresh objects.

Resources

David Guerrero is a system and network manager for the Boletin Oficial del Estado. He has been using Linux since the .98plNN days and now is playing with some Alpha-Linux boxes. When not working or studying, he likes to spend time with his love Yolanda, travel, play guitar and synths, or go out with his “colegas”. He can be reached at david@boe.es.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Creating a Web-based BBS, Part 2

Reuven M. Lerner

Issue #58, February 1999

Mr. Lerner continues to look at the bulletin board system, examining the code that works with individual messages.

Last month, I demonstrated how to build a small bulletin-board system (BBS) on the Web using Perl and a relational database. Such a bulletin board is another useful tool for bringing people together on a web site. This month, I will show you how to write several different programs, including ones that create and list the current messages.

Before continuing, let's review the two tables that contain the information in our BBS. I used SQL to define the tables, which means that while I wrote the database using MySQL, most of these definitions should work with other relational databases as well. The code to create these two tables is shown in Listing 1.

Listing 1. Table Creation Code

These two tables will enable a number of tricks to be performed with threads (i.e., message groups) and messages. Each message belongs to one thread. Each message (and thread) is contained in a single database row includes the author's name, her e-mail address, a subject heading and the text of the message.

Pointers to additional information about relational databases in general or MySQL in particular can be found in “Resources”, which also includes information about Perl's vendor-independent database interface (DBI).

Creating a Thread with Cookies

Last month I wrote a program, list-threads.pl, to list the threads (discussion topics) currently defined, and one to create a new thread, add-thread.pl.

However, I didn't provide an HTML form for use with add-thread.pl, because that form must be produced within a CGI program. The program that performs this function is add-thread-form.pl in Listing 2 in the archive file. Listings 2 through 5 are not printed here due to space considerations, but are available by anonymous download in the file ftp.linuxjournal.com/pub/lj/listings/issue58/3252.tgz.

Why use a CGI program rather than a static form? To be honest, the only reason is to provide a bit of functionality I particularly like. I order many items on the Web, often returning to the same vendor multiple times. I dislike having to enter my name and e-mail address every time I fill out an HTML form on the Web. I decided to make life a bit easier for people using our bulletin board system by automatically filling in the “name” and “email” fields with information the user had posted in a previous transaction.

If I were to use a templating system such as Embperl or ePerl, I could use a file that looks closer to standard HTML without burying the HTML inside of a CGI program. For a variety of reasons, including the fact that my web-space provider had not made mod_perl available as of this writing, I decided to use CGI programs rather than templates.

Regardless of whether CGI programs or templates are used, inserting a value from a previous form submission requires keeping track of state across HTTP transactions. HTTP is a stateless protocol; that is, each connection occurs without any memory from previous ones. How, then, can data be retrieved from a previous form submission?

The answer is HTTP cookies, a clever hack that has become a cornerstone of commerce and transaction on the Web. A cookie is a name,value pair, somewhat like a variable or an entry in a hash table. The cookie is stored by the user's browser, however, meaning that it is available across multiple transactions.

A site can set a cookie as part of an HTTP response, with a “Set-cookie” header. Whenever the user visits a site that previously set a cookie, it includes a “Cookie” header in its HTTP request. Thus, the cookie's value can be used to automatically fill in the “value” attribute of the “name” and “email” fields in the HTML form. When the user submits the form to create a new thread, the program sends headers that set the “name” and “email” cookies on the user's browser. The next time the user visits the site, those values are sent as part of the HTTP headers and can be retrieved and used within our program.

Perl's CGI.pm module allows us to easily work with cookies, using the “cookie” method. The following code is put in the form-creation program:

```
my $email = $query->cookie(-name => "email") ||
     "";
print "<TR>\n";
print "<TD>Your e-mail address</TD>\n";
print "<TD><input type=\"text\"
size=\"50\" ";
print "value=\"$email\"
name=\"email\"></TD>\n";
print "</TR>\n\n";
```

which assigns **$email** the value of the “email” cookie or the empty string. The empty string could be ignored, since Perl automatically assigns the empty string to a variable the first time its value is retrieved. However, this would produce a warning message, since the program would be using the value of an undefined variable. It is safest to assign the empty string when possible.

The value of the text field is set to the current value of the cookie or the empty string; that is, either the user's e-mail address or nothing at all. A similar method is used for the user's name, so that she doesn't have to enter her name multiple times.

The form is submitted to add-thread.pl, which we examined last month. That program uses the elements of the submitted HTML form to create an SQL query that inserts an appropriate row into the ATFThreads table. Because we have defined the ID column of ATFThreads with the AUTO_INCREMENT attribute, we can be sure every thread will have its own automatically generated ID number that we can reference in our programs.

When the form is submitted, our CGI program creates two new cookies, one named “email” and another named “name”. We can then retrieve the values with CGI.pm's “cookie” method, as demonstrated above. Creating the cookies is almost as easy as retrieving them:

```
my $namecookie = $query->cookie(-name => "name",
  -value => $query->param("name"),
  -expires => "+1y");
my $emailcookie = $query->cookie(-name => "email",
  -value => $query->param("email"),
  -expires => "+1y");
```

Once we create **$namecookie** and **$emailcookie**, we can send them to the user's browser, thus setting the cookie values, by incorporating them into the HTTP header:

```
print $query->header(-type => "text/html",
 -cookie => [$namecookie, $emailcookie]);
```

Since both cookies are set to expire one year (+1y) after they are created, the user's browser should continue to send the name and e-mail address whenever visiting the site in the future.

Working with Messages

Now that we have seen how the underlying database system will work for threads, we need to begin working with the actual messages. Because this is a simple system, we'll look only at posting a new message to a thread and viewing the contents of a thread.

In many ways, posting a new message to a thread is similar to creating a new thread. In both cases, the user's name and e-mail address are requested. In both cases, the date and time at which the thread was created is recorded, and the user can enter a title and a message body.

The major difference between messages and threads is that each message must be associated with a thread. This association is used to create the illusion that the messages are stored separately, when in fact they are all stored in the same table (ATFMessages). Users, however, will be able to see only a single “thread-wise slice” at a time.

Just as I used a program to create the thread-adding form, I will use a CGI program to create the message-posting form, called post-comment-form.pl (Listing 3 in the archive file). This form will submit its contents to post-comment.pl (Listing 4 in the archive file).

I will ensure that each message is associated with a thread by putting a selection list inside of the form. Each option in the selection list will be identified internally with the ID code for the thread in question and will display the subject line.

In order for this list to reflect the current status of the database, a database query is done and the results are displayed in the form. The query is set by:

```
my $sql =
"SELECT id,subject FROM ATFThreads ORDER BY subject";
```

and then executed, iterating through each **id,subject** pair. Each pair is inserted into an **<option>** tag, as we can see:

```
while (my @row = $sth->fetchrow)
{
print "<option value=\"$row[0]\" ";
print " selected " if ($thread_id == $row[0]);
print ">$row[1]\n";
}
```

The standard DBI **$sth->fetchrow** method is used to return the next row from the SELECT query. When no more rows remain to be retrieved, **$sth->fetchrow** returns false, which ends the **while** loop.

Also notice how a particular thread's subject can be selected by comparing its **$thread_id** with **$row[0]**. **$thread_id** is set to the value of the query string, which can be loosely defined as “anything following the question mark in a URL”. The line:

```
my $thread_id = $query->param("keywords") || 0;
```

causes CGI.pm to automatically assign the parameter **keywords** to the value of the query string. If the user invokes the program with http://www.lerner.co.il/cgi-bin/post-comment-form.pl?5, then **$thread_id** will be assigned the value 5. If the query string is not assigned, the value is left at 0, in which case no default thread is selected.

Posting the Message

When the HTML form is submitted to post-message.pl (Listing 4 in the archive file), the form elements are used to insert a new row into ATFMessages. As I indicated above, post-message.pl is not very different from add-thread.pl, except that it stores a thread ID number along with all the other information:

```
my $sql = "INSERT INTO ATFMessages ";
$sql .= "(thread,date,author,email,subject,text)";
$sql .= "VALUES ";
$sql .= "($thread_id,NOW(),$name,$email,$subject,$text)";
```

The variable values can be inserted without surrounding them by quotes, because the standard **$dbh->quote** method was used. I discovered this method only recently and continue to be amazed that I was ever able to survive without it. Simultaneously, the form elements are retrieved and quoted appropriately in the following lines of code:

```
my $name = $dbh->quote($query->param("name"));
my $email = $dbh->quote($query->param("email"));
my $thread_id = $dbh->quote($query->param
("thread"));
my $subject = $dbh->quote($query->param
("subject"));
my $text = $dbh->quote($query->param("text"));
```

Once this is done, the above SQL query will INSERT a new row. We tell the user that the new message has been added and produce a menu bar with a number of options.

Believe it or not, these two short programs are all that is needed to insert a message into the database and thus into our BBS.

Viewing a Thread

At this point, the functionality is close to complete. All that remains to be done is to create view-thread.pl (Listing 5 in the archive file), which allows us to look at the current contents of a thread.

For this program to work, a single argument must be passed in the query string to identify the thread. To retrieve this value, use the **keywords** HTML form element that CGI.pm creates:

```
my $thread_id = $query->param("keywords");
```

Once **$thread_id** is assigned, I can retrieve the appropriate information from the tables about that thread. Indeed, two separate queries are done: one from ATFThreads and a second from ATFMessages. (I could have combined the queries into a single large SELECT statement, but I chose to keep them separate.)

Early on, I decided to print the date and time of the user's posting along with the text of the posting. Given the DATETIME data type, how can we retrieve the date and time in an intelligent way? MySQL provides a DATE_FORMAT function which takes the value from a column and writes the contents using a specified format.

To make life easier, I actually retrieve the same “date” column twice, once for the date and again for the time. This allows literal characters to be inserted between the date and time without having to worry about possible misinterpretation:

```
$sql = "SELECT id, DATE_FORMAT(date,
   \"%W, %d %b %Y\"), ";
$sql .= "DATE_FORMAT(date, \"%h:%i %p\"), ";
$sql .= "author, email, subject, text FROM ATFMessages ";
$sql .= "WHERE thread = $thread_id ORDER BY date desc";
```

DATE_FORMAT takes two arguments: the name of the column to retrieve and a set of codes (in the style of C's **printf** statement) indicating the values to use.

Once this query is executed, the code iterates through the results, printing the messages as they come—from newest to oldest. They will come in that order because of the ORDER BY clause in the SELECT statement. Allowing the database to do our dirty work for us means we can print all of the messages in a thread with just the following short loop:

```
while (my @row = $sth->fetchrow)
{
my ($id, $date, $time, $author, $email, $subject,
   $text) = @row;
print "<a name=\"$id\"><B>$subject</B>, ";
print "by <a href=\"mailto:$email\">$author</a> ";
```

```
   print "on $date at $time</P*gt;\n";
   print "<blockquote>$text</blockquote>\n\n";
   }
```

Summary

The basic BBS software is now finished. It can create threads, add a message to a thread and view the messages within a thread. If nothing else, this project shows how powerful a set of database tables can be when paired with some CGI programs.

Perhaps this system is too basic; it is lacking some functions that any good bulletin board (or any good web application) should handle. For instance, it would be nice to include the ability to search through the posted messages for a text string or regular expression to find messages relevant to a particular topic.

It would also be useful to provide some administrative functions, unavailable to the public at large, for handling things at a relatively high level. For example, we might want to reserve the ability to delete messages that are offensive or unrelated to the topic. We might even want to give this ability to certain other users, giving them “deputy moderator” status.

Next month, we will see how to provide these functions by adding just one or two new CGI programs. Meanwhile, you can see these programs in action at http://www.lerner.co.il/atf/, where the “At the Forge” BBS is already in use.

Resources

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

Archive Index Issue Table of Contents

Advanced search

Focus on Software

David A. Bandel

Issue #58, February 1999

gtkcookie, guitar, gentoo and more.

Welcome to the second installment of my look at software packages worth downloading. I always find something good on the Internet. A lot of developers have been quite busy and are turning out some excellent software. The hot development library seems to be GTK+, and with so many novice Linux users who are not command-line oriented, it is a good thing.

gtkcookie:

http://www.110.net/~pq1036/

Do a lot of web surfing? Notice how many sites want to leave little droppings called “cookies” on your system? So many, in fact, I added a hard drive to store them all. Okay, I actually needed more space to download and compile programs like this one. It is a slick, easy way to view, edit and delete cookies. Frankly, I would rather eat them. Required libraries are gtk-1.0.6, Xi, Xext, X11, m and glibc.

guitar:

http://disq.bir.net.tr/guitar/

guitar is another of those utilities you love to have. I like this wrapper for tar despite the fact that I am quite comfortable on a command line. **guitar** lets you see what is inside a tar or gzipped tar file, read the text files inside, and extract the one you want. It also lets you create a directory for storing the files. This version does not create tar files, but I'll bet future versions will. Required libraries are gtk-1.0.6, Xi, Xext, X11, m and glibc.

gentoo:

http://www.obsession.se/gentoo/

gentoo is yet another file manager. It looks simple, but hides a lot of complexity. It will copy, move, and do other things as well. A three-row button bar on the bottom can be configured as you see fit. In fact, the most complicated part of this program seems to be figuring out how to fill up all the empty buttons rather then actually doing it. The configuration box takes a cue from other recent systems that use tabs across the top of the box to change to various configuration pages. Once it is configured (the default configuration is actually quite good), it is easy to use. Required libraries are gtk-1.0.6, Xext, X11, m and glibc.

slashes.pl:

http://members.xoom.com/alexsh/slashes/

slashes.pl is a small Perl script that all the "dotheads" in the audience will like. It is a way to keep the *Slashdot.org* headlines on your screen while having your web browser open to it all day long, just so you can press reload. This program allows you to see the headlines whenever you wish, just in case a news article appears that you would like to read. While I had trouble with the "Read" button even after setting the line to point to my Netscape binary (it would crash), I suspect I just need to reread the script and find out what I am doing wrong for my setup. Required libraries are Perl, gtk-1.0.6 and the libgtk-perl module.

The Gaby Address Book of Yesterday:

http://www.multimania.com/fpeters/gaby/

Gaby is a deceptively simple address book. It carries all the latest fields for those needing to keep up with more than just phone and fax numbers. Space is reserved for URL, e-mail address and more. What it lacks is a way to connect to a database engine like MySQL. The flat-file method works well for individuals, but is not practical for a site where multiple users may have years of contacts listed. This one is great for single users and has the potential for larger sites if it can connect to a database engine. Required libraries are gtk-1.0.6, Xext, X11, m and glibc.

The Amazing Anagram Thingie:

http://www.vis.colostat.edu/~scriven/anagrammer.php3/

This is a command line “game” to help you with those pesky anagram puzzles—very fast, very simple and easy to use. A phrase of any length will have this one scrolling anagram solutions off the screen for a long time. It is a great game for those puzzled by anagrams. Required library is glibc.

Ministry of Truth:

http://tomato.nvgc.vt.edu/~hroberts/mot/

This particular package has been out for a few months, has earned a place in my own system and will soon occupy an internal web site where I regularly work. Sometimes keeping track of jobs is difficult. This program makes it a breeze. It can track most users, software, equipment, jobs and people associated with them. While it requires MySQL, it handles everything once MySQL is installed and configured. For those requiring assistance, a (very low volume) user mail list has been set up. Required libraries are Apache with the php3 module compiled with MySQL, and MySQL.

Still lots more good packages out there. See you next month.

David A. Bandel (dbandel@ix.netcom.com) is a Computer Network Consultant specializing in Linux. When he's not working, he can be found hacking his own system or enjoying the view of Seattle from an airplane.

Archive Index Issue Table of Contents

Advanced search

Letters to the Editor

Various

Issue #58, February 1999

Readers sound off.

Applixware 4.4.1

I recently purchased Applixware 4.4.1. My reason for purchasing it was so that I could interoperate with co-workers who run MS Office. Applixware has the best MS Office import filters I have seen, but they are not quite perfect. I found that while text and tables seem to import well, figures are another story. So far, I have only tried importing a few MS Word 7.0 documents, but in general, the figures fail to convert. I recognize the complexity of the conversion task, and I applaud Applix for doing as well as it does. I hope the filters continue to mature, so that I can finally dump Windows 95 off my machine and use Linux exclusively.

—Steve Falco sfalco@worldnet.att.net

Erratum in *LJ* February 1998

I know that it is late to correct an erratum in the February issue; nevertheless, I think *Linux Journal* is a magazine to read and save for later use, so even such a late correction could be of some benefit.

The error is in Listing 5 of the article “Attaching Files to Forms”, in the column “At the Forge” on page 93. This listing should contain the following line between lines 2 and 3:

```
no strict "refs";
```

If this line is not present, the Perl interpreter will abort the script as soon as the variable reference **$userfile** is used for write. The uploaded file will be created but not written, i.e., it will have zero length.

—Aldo Mozzi medico.red@interbusiness.it

Answer to User's Question

In the November *LJ*, "Best of Technical Support" had a question regarding installing Linux (specifically X) on an IBM Thinkpad 365. There is an excellent article regarding this in *Linux Gazette*, http://www.ssc.com/lg/issue21/notebook.html.

I was able to get X working on Thinkpad using this guide, and the author, Sam Trenholme, was extremely helpful in getting me over a few problems when I contacted him via e-mail.

—Nate Dutra nate@the-wall.net

Eiffel, Design by Contract

Your fine web site, http://www.linuxresources.com is usually the first thing I check on weekdays when I boot up. You have helped me greatly in learning about Linux.

Thank you very much for reporting on Dr. Meyer, Eiffel and Design by Contract. Here are three open-source Eiffel and Design by Contract resources:

SmallEiffel, the official GNU Eiffel: http://www.loria.fr/projets/SmallEiffel/

TOM, a new GPL/LGPL OO language with multiple inheritance and Design by Contract traits: http://gerbil.org/tom/

A page I edit: http://www.newhoo.com/Computers/Programming_Languages/Eiffel/

—Jerry Fass fass@pitnet.net

Linux Installation and the Open Source Process

Last week I decided to totally change over to Linux after reading the latest horror story about Windows. Apparently, people connected to the Internet on W-95-98 can be snooped on simply by typing their IP number and a few backslashes. Suddenly, their whole system appears in the other person's MS Explorer window.

So here I am, scurrying to get a faster motherboard and a bigger hard drive. I am all enthusiastic about the new KDE and GNOME projects, yet when I read the installation manuals—S.u.S.E. or Red Hat—I am horrified by how much

totally new and alien system configuration is needed for a new Linux user. The problem is exemplified by the marvelous advertising flyer sent out by S.u.S.E. for its 5.2 release. It seduces the user with a DOS-Windows box packed with a lifetime accumulation of precious utilities. It implies that UNIX clones for these treasures (and more) effortlessly await on the new Linux user's desktop. The actual case is that this happens only after endless installations and configurations.

The open-source concept has been much in the news lately, yet it seems that these installation processes are the one place where the open-source environment is not used to evolve solutions to these problems. Rather, it is the special province of the private bundling companies. Regardless of whether they put their products in the public domain, they are not developed in the open. People seem to believe that the greed of these companies will produce the fastest results, but I have not seen any miracles yet. Perhaps there are installation projects underway in the open-source community. If so, nothing could be more critical to the advancement of Linux.

We often hear people crowing in *LJ* that some huge corporation or the defense department is now more efficient because of Linux. I am much more impressed by ease of use by the ordinary person. If *LJ* were to make open-source installation projects a continuing focus of articles, it could do an incredible service to the evolution and spread of Linux.

—David Briars dbriars@sover.net

Easier installation seems to be what everyone is asking for these days. Red Hat is working on it with LinuxConf, and Caldera with COAS (see article in this issue) —Editor

Happy Hacking Keyboard Price

The correct price for the Happy Hacking Keyboard at the time of the article was $159, not $189. Today I cut the price again from $159 to $139 as a standard price. Please let your readers know.

—Ted Abe, PFU America, Inc. tetsu@pfuca.com

BTS Correction

In the November 1998 “Best of Technical Support”, weird things have been happening to Eric Benoit's system. **su** misbehaves, as do **man** and **less**. Scott Maxwell thinks it may be terminal options, but the answer lies with /dev/tty. My bet is that somehow, something has changed the permissions on it and it is no longer readable. Just type:

```
chmod u+rw /dev/tty
```

and all will be well. You should probably make sure it is writable while you are at it.

I guess most programs that decide to access /dev/tty to talk to a real user never consider the possibility that they do not have permission to do so.

In general, processes inherit an open file handle to /dev/tty from their parent as stdin, stdout and stderr. The process that opens them in the first place is **login** which is running as root, so it has no problems. Either that or the top-level program opens a particular real or virtual terminal using a device name which will have the permission bits set differently.

—Adrian Pronk apronk@sangacorp.com

Linux Journal and Red Hat

I have been a subscriber for about one year and I really like the articles. I am happy to see that Linux is stealing the spotlight from Microsoft.

I first got into Linux in 1996 after working with the HP-UX workstation on a job and realizing that you can connect to the Net with any OS, not just Macintosh, Windows 95 or Windows 3.x.

Red Hat is a good distribution and I have been running 5.0 for a year without any problems. I plan to get 5.2 soon.

For all you newbies, I recommend *Linux for Dummies*, *UNIX for Dummies* and *Teach Yourself Linux in 24 Hours*. These books include the operating system and some applications. I would recommend Red Hat to anyone.

—Fred Nance fnance@eclaim.com

VFS Error Messages

Regarding December's “Best of Technical Support”, “VFS Error Messages”: another reason for the failure to mount the root partition during boot is that the root file system was compiled as a module when the kernel was built, rather than compiled into the kernel itself. When the root file system is first mounted during the boot process, no modules are loaded, and modules cannot be loaded until the root file system is mounted. If the file system driver is a module, then the kernel cannot mount it—so it panics.

—Rob Singleton single@nortelnetworks.com

Real Life Business Story

As a system integration tool, Linux has allowed us to prepare custom network file servers which can do the following:

- Provide complete web-server services (Apache).
- Provide Internet connectivity for many users on the local LAN (IP-Masquerade).
- Provide file and printer services for Windows/DOS users (Samba).
- Provide file and printer services for Netware users (MARS_NWE, NCPFS).
- Provide complete internal/external e-mail services (Sendmail).
- Provide inexpensive terminals in both graphical and text-based platforms with the X Window System which can be connected in a variety of ways (Ethernet, serial, etc.).
- Provide complete point-to-point protocol (PPP) implementation for routing and other remote-oriented operations.
- Provide a fully scalable system that can grow with the company.

All of the above have been thoroughly tested and implemented. We could not be happier with the performance and continued development of this OS.

—Larry Rivera larrydog@coqui.net

Review of *Learning the Bash Shell*

In the review of *Learning the Bash Shell, Second Edition* (December 1998), Bob van der Poel points out that the book examples available by FTP are from the first edition and that the correct ones might be available by the time the review was printed. That should now be true, as I asked the publisher to correct this mistake in October.

—Cameron Newham, Author cam@sspl.demon.co.uk

Re: CIDR

In the December issue, there is a misprint in David A. Bandel's article “CIDR: A Prescription for Shortness of Address Space”. On page 26, under the CIDR heading, second paragraph, it states:

> For a Class C address, this default subnet is 24 bytes long, so putting all ones in the first 24 bytes and zeroes in the rest, we have 255.255.255.0.

I think this should instead read 24 bits, rather than bytes, since each octet is composed of 8 bits, which gives 4 total bytes. Just wanted to bring this to your attention. Great article!

—Bob Cummings bob@cter.eng.uab.edu

Mea culpa—you are 100% correct. Thank you for pointing out that little lapse of attention on my part. Guess I need to re-read everything thrice, because I read it twice and missed it both times. —dbandel@ix.netcom.com

Archive Index Issue Table of Contents

Advanced search

More Letters to the Editor

These letters were not printed in the magazine and appear here unedited.

BTS: Re: Netscape Mail article Dec 1998

I am replying about the article in the December 1998 issue of *Linux Journal*, in Best of Technical Support, about Bill's question on Netscape Mail, Demon Internet and Red Hat 5.0:

Demon Internet are unusual as an ISP in terms of providing a full SMTP feed to you, treating you like a “real” Internet host, rather than relying on POP3. It is only recently that they have started providing a POP3 service for collection of mail.

The reason you are getting some mail in Netscape and some in Sendmail is that Demon open an SMTP socket to you as soon as you log on, and start sending mail to you. If you also connect via POP3, some will come down via each method.

The solution for Red Hat 5.0 is to turn off Sendmail, using the command 'chkconfig sendmail off' - this will prevent Sendmail starting, thus preventing you from having anything listening for an SMTP connection, preventing Demon from connecting to you. All your mail will stay on the Demon side, waiting for you to pick it up with your POP3 client.

Sorry for being so wordy. Please feel free to edit this down a bit if you want to publish any of it.

Hope this helps,
—Peter Denison
peterd@pnd-pc.demon.co.uk

Re: Netscape Mail

> Peter Struijk writes in *LJ*:
> Sendmail is most likely sending out mail with your full host name. When a recipient replies to a message you sent from elm, the reply is sent directly to your computer and not to your mailbox at your ISP.

Peter, you have inadvertently misled Bill here. Demon Internet, a large UK service provider, have an uncommon setup for email.

Your actual host name 'mirrim.demon.co.uk' is the same as in your email address. Demon users have an unlimited number of email address in the form <user>@<hostname>.demon.co.uk. Demon set the MX records for the dial-in hosts to point the their own mail servers, so mail is never delivered directly from outside hosts to the clients' machine by SMTP.

However, Demon _do_ deliver mail by SMTP to client machines. This is their preferred method of mail delivery, and their 'Turnpike' software for Windows accepts mail in this way, and maintains multiple mailboxes.

Demon have recently started to offer an additional service, where mail for dialup hosts can be fetched by POP3 instead of by SMTP. Naturally this conflicts with the SMTP delivery, so some messages are delivered by each method. If you intend to use POP3 for mail delivery, then you should disable the SMTP daemon on your machine.

A simple way to do this is on a Red Hat box is to type (as root of course):

```
        /sbin/chkconfig --del sendmail
```

It's best to set up sendmail so that it will forward mail to Demon's mail hosts if it can't deliver it directly. That way, they can keep trying for you, even after you've disconnected.

I can't tell you how, though, because I don't use sendmail - I use Exim (http://www.exim.org/) because it's more secure, faster and easier to configure.

I would suggest using a local mail service, though, and allowing Demon to deliver mail by SMTP. Then you can 'POP' it from your own machine. That way, you can easily use more than one of your unlimited mail addresses. Even if it's only ever going to be used by yourself, you might find it useful to have multiple addresses going into different mailboxes.

Demon sometimes takes a minute or two to deliver mail after you dial up. If you intend to dial up just for the purpose of fetching mail, and not doing anything else at all, then this might annoy you.

In this case, you could use a modified version of the 'fetchmail' POP client, which uses Demon's POP3 extensions to find out the intended recipient of the mail and deliver it to the right place. The patch to fetchmail ... isn't on my web site ATM for some reason. I'll mail it to you if you ask for it, and put it up for FTP somewhere.

Feel free to get in touch if you have any problems with it, or if you want further clarification.
—David Woodhouse
David.Woodhouse@mvhi.com

Midnight Commander

As an utter Linux newbie, it took me a month to find MC (http://www.gnome.org/mc/). Now that it's too late of course there it is on on my Red Hat distribution; for all I know, all I had to do was type MC....

Whatever, (1.) I wanted to thank from the bottom of my heart the creators of MC; I just can't express the feeling of freedom and release a little tree and file display produces; all your karmas are immensely enhanced! (2.) Distributions really ought to consider emphasizing this thing more. At this moment, Linux is poised for greatness, or at least to significantly annoy Bill Gates, and providing this kind of obvious file/directory support to new single-machine users like myself is an obvious way to encourage Linux use and distribution.

But again: thank you, MC creators.
—James G. Owen
71121.625@compuserve.com

Subnetting Table

The article about CIDR in issue 56 by David A. Bandel was very interesting. Sometime ago I created a subnetting table that illustrates these concepts.

It's available as PDF at

http://www.access.ch/ml/software/subnetting/subnetting-table.pdf

You might want to publish the link for your readers...
—Marc Liyanage
webmaster2@access.ch

Funny attitude

Please accept this letter as it is, and not as an insult to your magazine.

Reading a couple of issues, I have noticed a weird attitude of the people writing articles. Everybody is against Microsoft, but in a very childish manner. Of course, I agree that Microsoft products are not the best, they are not even decent, but expressions like “In the continuing battle with the Evil Empire, I have recently reduced my dependence on ”Them“ blah-blah”(issue 47 page 64) or the Fry Electronics display window (issue 54 same page). Let's get serious, people. It's not the way you can fight software monopoly. Serious people don't bark against Microsoft, serious people (Linus, Alan and many more) write good and free software. Serious people don't publish articles about “Yes! In Zair we can make a full elephant migration survey using Linux!”. No. Serious things like that is made using more powerful software, powerful than Linux. Linux is a good OS, free, but it's still at its beginning. I've just got pissed off of that “Linux propaganda” many magazines are doing. You know, that propaganda makes Linus look like Lenin in Linnuxism (communism). I hate these extremist opinions, and please be more careful in the future.

Linux Journal is still one of the best Linux magazines and I hope it will remain so. Above Linux (although it's free and makes a lot of people earning lots of money paying nothing instead) there is democracy and the freedom of choice.

I am sure this will not be published, but I am satisfied that what I had to speak was heard where it was supposed to.
—Stefan Laudat
stefan@art.ro

IDG/LinuxWorld Expo alienates speakers?

The following is a quote from Alan Cox's Dec. 11th. dairy:

> (http://www.linux.org.uk/diary) entry:
> I also had some non fun mail from the LinuxWorld people (ie IDG) when I asked them to clarify arrangements for speakers expenses. Answer “we wont be paying any”. Thats one less speaker. I know three other speakers who will also probably be dropping out and no doubt more will follow when they discover this.

> Now in my case sure I can probably extract the money from someone but there is a principle at stake. Many Linux hackers are in it for fun and don't get paid for it. A conference whose financial greed extends to excluding all the non commercial Linux hackers is wrong. It may be how those dreadful non technical all gloss networking/ windows shows run but its not how a technical conference should be run. It's not how other Linux events are run and its not how Usenix is run.

> I may be a member of the small club of Linux people who can get funding to attend and speak at such an event but I want no part in it.

I am one of the organizers of the Atlanta Linux showcase and have been since it was started over three years ago. I think things like this are an important distinction between a Linux trade show whose sole purpose seems to be to take advantage of the Linux community in order to generate trade show revenue and ALS, which is as close as we can get to an Open Source type of Linux trade show. ALS is put on by a not-for-profit corporation made up of volunteers from the Atlanta Linux Enthusiasts user group and we have always covered travel expenses for people who were willing to take time out of their busy schedules to come and speak at ALS.
I think it is important for the Linux community to realize the possible consequences that can happen when pure commercial interests intersect with the Open Source community. Some times becoming the hot press topic and being seen as the next Microsoft competitor may not always have pleasant results.

—Steven A. DuChene
sad@ale.org

ATI video cards

Some time ago I bought an ATI all-in-wonder. This card was support by Xfree86 on the video side but not the TV tuner side. When someone, I can't remember who, tried to enlist ATI for assistance to write a driver for the tuner, they declined and have declined in spite of a large assault from many Linux users.

All that said I was distressed when I read in the December 1998 issue #56 in the 'Best of Technical Support' in a reply to the email entitled 'Networking' *LJ* printing a reply which plugged ATI products.

If vendors do not support Linux then the Linux community should not support same.

As far as I know ATI is one of these vendors.

Thanks,
—Gene Imes
gene@ozob.net

Linux on Macintosh

After years of using Wintel equipment and successfully installing Red Hat Linux on a 90MHz Pentium with 32MB of RAM and a 1.2 Gigabyte disk I purchased an Apple Macintosh G3. I use Macs at work and I'm familiar with them. It wasn't until I started using a Mac exclusively that I realized the I much preferred the elegance of the interface and the integration of the software from third parties.

I have an interest in learning UNIX and the obvious course of action was to learn Linux, And even though I have a Mac now I still have a great interest in learning Linux. Having heard a little about Linux on Mac equipment I decided to look for myself to see what was available.

When I visited your web site and saw that the January issue would have an article regarding Linux on a Mac. I was really excited and could hardly wait for it to arrive in the mail.

That day has arrived.

I am very disappointed.

I am currently preparing to install LinuxPPC on my PowerMac G3. With a 4 Gigabyte disk I can partition it and run both the Mac OS and Linux peacefully and actually communicate between the two partitions. I would like to direct your attention to http://www.linuxppc.org. At that site you will find much more up-to -date information regarding Linux and Macintosh.

I would also like to point out MKLinux,to my understanding, was developed by Apple for the 68k chip. Apple is no longer developing MKLinux.

A new distribution is in development called Yellow Dog.

After I found all the information available regarding Linux on Macs I started looking though back issues of *Linux Journal* to see if there was anything I had missed on the subject. Unfortunately there wasn't.

For inside, blood and guts, info on Apple Macintosh architecture and Motorola chip architecture I would like to direct you to the Inside Macintosh series of books. They are published by Addison Wesley with information provided by Apple. The series has been available for at least four years and is regularly updated. Motorola has information on their chip architecture in the Motorola user's manuals.
—Brian Vawter
bvawter@psnw.com

> Sorry you were disappointed in Alan's article but as you pointed out, it was meant only as a discussion of porting the kernel not a HOWTO. *Linux Journal* has had other articles on Linux and the Macintosh: “How to Build a Mac” in issue 19, “MkLinux: Linux Comes to the Power Macintosh” in issue 31, “Linux? On the Macintosh? With Mach?” in issue 37, and “Netatalk, Linux and the Macintosh” in issue 45. We also have one promised but not yet delivered for future publication. —Editor

---

Stone Age hardware in PC

In “Linux for Macintosh 68K Port”, Alan Cox complains about the Mac's Stone Age hardware design. The PC also has a little Stone Age hardware.

The Intel 8259 interrupt controller was introduced as a peripheral for the Intel 8080 eight-bit microprocessor. The 8259A added a mode for compatibility with the 8088 and 8086 processors.

The 8259A has a mode where the interrupt vector is returned as three bytes. This mode is not used with x86 processors, so I do not know if modern 8259A emulations include it.

The three-byte mode emits an 8080 machine-language CALL instruction. The 8080 executes this instruction to get to the interrupt service routine.
—Peter Traneus Anderson
peteand@vitech.com

Choosing between VI and World Domination

One again we have had the survey about favourite editors and once again VI emerged victor and if I judge by what happened last year you are receiving loads of e-mail of VI partisans happy of the outcome.

But. It is not to an old statician like me you will teach him about sample bias and about the problem of spontaneous votes who tend to over represent the most vocal and activist group.

LJ readership is only a tiny fraction of Linux population and tends to be more extremist and Unixically correct than your average Linuxer.

More importantly you are not asking to the people who saw Unix and hated it. Each time I meet one of them I ask him. So far the first thing they mentioned has always been the same: “Unix's awful editor”. They were not speaking about Emacs :-).

When did Unix reach world domination? NEVER. The crown went directly from mainframes to DOS. And blindly following the traditions of a system who never won is a sure way to defeat. A system who got an awful reputation and lost many potential followers in no small measure thanks to VI.

Because Linux will reach world domination the day people like lawyers or writers will use it. And lawyers will NEVER use VI.

So, VI is the editor you will find in every UNIX? Linux will crush proprietary Unixes, the sooner, the better. Who cares about their editor?

It is time we begin designing things the Linux way and stop caring about Unix.
—Jean Francois Martinez
jfm2@club-internet.fr

Corel WP for Linux.. Very nice :-)

Just like to tell you how easy it was to install, configure and run WP for Linux. I have worked with Linux for four years and this was one of the easiest apps I have ever installed.

So far I am very impressed.. I will forward this e-mail along to *Linux Journal*..

If you are looking for system config feed back..

System:
FIC 2013 Mother board
AMD K6 350
128 MB ram
6 GB HDD

AGP 8 MB Matrox millennium II
17" AOC @ 1024 x 768

Kernel 2.0.36
Xwindows Xfree86 3.3.3

I hope to see more of your apps ported to Linux in the future

We are also selling Linux gaming computers and are considering an Office computer system.
—Doug
fixxxer@accessgate.net

Macsyma - yet another company that ported its product to Linux

Your review on Mathematica 3.0 for Linux by Patrick Galbraith (*LJ*, December 1998, page 75) gave me an idea that one could mention another big player from the Computer Algebra Software field, namely Macsyma, that just recently ported its product on Linux. The third one of the Big 3 from this field (MAPLE) has been available for quite some time.

You can find more about Macsyma, and its Linux version, at http://www.macsyma.com/

Let me just add that I find Macsyma to be extremely good. I suggested to some people at Macsyma last year to make it available for Linux users (I am not pretending that my suggestion was either first or the only one). My argument to them was that I see some similarity between Linux and Macsyma, that I believe that people using one of them will easily get to like the other one. In particular, I mentioned neatness, precision and “a mathematical carefulness” of Macsyma, together with its affordability. Namely, the price of Macsyma for Linux is around $200, versus $1,495 (according to the above mentioned *LJ* article). I did not check the price for Maple, but am pretty much sure that it is over $600. On the other side, Macsyma scored far better on some comprehensive independent tests (see the Macsyma's home page for the details), and it seems to be easier to learn for beginners.

Let me use this opportunity to add another idea. I have been subscribed to *LJ* for about a year, and using Linux (Slackware 3.0) for close to 2 years. I enjoy both, Linux and the *LJ*, very much. Many articles are over my head, but many are just right; and I read everything anyway. This particular January 1999 issue has so many articles that I found to be really great. I filled in a little feedback to 2-3 of them just a few minutes ago, but then got kind of tired of typing the same thing over and over again (I apologize to the other authors for my laziness). So here is my idea: Will it be possible to make link where one could pick several articles at once to send his feedback?

Thank you very much for your kind consideration, and for the great job that *LJ* has been doing.

—Milan Lukic
lukic@vulcan.acs.uwosh.edu

> When *Linux Journal Interactive* gets up and going (soon, we hope). The response forms will go away —instead there will be discussion groups for each article. This will still be done on an article by article basis though, so I'll submit your idea for an “in general” group to our tech department. —Editor

---

MkLinux, LinuxPPC, Turbo and Debian

I realize the subject of my e-mail might seem weird so here is a clarification. The question: How many versions of Linux are there for the Power PC based Macs? The answer: 4, The subject of this e-mail (as far as I know).

This is not meant as a mean type of letter I just want you to know. The computer industry as a whole seems to be at the threshold of a new rule with Microsoft being dumped left and right. Where are these people going, Linux? This is a great opportunity for opening of standards and more freedom. As a Macintosh user I have noticed a big turn around in peoples perception of Apple, there machines, and there OS (I am a Mac Rep at CompUSA). I realize that most people who use Linux are using it on a Wintel style machine but as we all know, with Linux, this doesn't matter. Linux is an OS that really doesn't care about the machine that it is on. With LinuxPPC R5 able to run on iMac's ,and any PowerPC for that matter, also there is info out on the web that suggests that Apple will bundle a version of Linux on some of there machines. Would you please do some more articles that have some type of Mac connection so more Mac users can realize the power of Linux. Maybe an article a month and or a Special Issue that has info for Mac people so they can start using Linux also. For instance which of the 4 listed distributions should people use for there need? Why should they use Linux? Where can they locate software they will need?

I personally have a tri-boot of MkLinux DR3, LinuxPPC R4, and Mac OS 8.5.1. I have been using Linux for about 6 months and have gone through most of the newbie problems and still go through them. I will probably get a subscription to your magazine because I have been impressed by the depth of knowledge your articles present every month. Thanks for reading and for all the new and old Mac based Linux users we hope to see a new column soon.
—Chad J. Adelman
casl4419@mindspring.com

BTS

On page 61 of the Jan99 issue of *Linux Journal*, you raise the question of how to allow an ordinary user permission to shutdown the system, and the reply is that the user must use Alt-Ctrl-Del.

I have learned an alternate solution (I cannot claim credit for figuring out this one myself, though). If you set the UID bit on /sbin/shutdown, then ANY user can use /sbin/shutdown. That's the catch—it allows any user at all permission to execute /sbin/shutdown. As root, type: chmod 4766 /sbin/shutdown

This also allows remote shutdowns: rsh machinename /sbin/shutdown -h now (or -r now or whatever options you want).
—Capt Christopher A. Bohn
cbohn@afit.af.mil

Re: BTS Shutting Down

In the Jan. 1999 Best of Technical Support in the *Linux Journal*, Thomas Okon asked about a utility to allow a user to shutdown the system without having to login as root. The solution is at sunsite as ftp://sunsite.unc.edu/pub/Linux/system/admin/su/usershutdown-1.1.tar.gz
—Andy Holder
aholder@bellsouth.net

BTS: GNU wget for updating web site

Re. the question “Updating Web Site” in the Jan 1999 *Linux Journal*, p. 61 ...

Haven't tried the mirror package - might be good, but you can also use GNU wget (prep.ai.mit.edu). Here is the script I use to keep the University of Maryland LUG's Slackware mirror up-to-date. “Crude but effective”.
—Judah Milgram
milgram@eng.umd.edu

Consistency vital to growth of Linux

When designing systems, or writing software, consistency, not perfection is the goal. Perfection can not be achieved, and too much time is spent trying to achieve it. The next best alternative is consistency. With consistency, even poor designs can be used. Examine for a moment the most popular OS for PC's. When I first started using it, I disliked it. I still dislike it, but have observed its growth. It did not grow because it is great, it didn't even grow because it is good. It grew because it is consistent (and of course marketing - people love someone that can make used motor oil sound like a health drink).

I first tried Linux after I left my personal computer (S-100 CP/M) behind and bought my first PC. It came with with a crashing OS with a defective windowing system, but I could edit code with it and if I didn't make the names of the files too long, I could move files back and forth to work.

One day, while at the computer store I noticed a CD with a distribution of Linux. I am happy to say I bought it.

I installed Linux and used LILO to dual boot. I liked Linux because it was very close to the OS I used at work. I found myself using the crash OS less (I needed it to run FrameMaker) until, finally I had to upgrade to undertake a challenging project, writing a MUMPS to C++ translator for a program that was over 0.5M lines of code. I needed a system that was dependable, so I upgraded to a 486, 128M of memory, 8GB of disk, a dat tape drive, and the latest Linux distribution I could find.

To test the system, I ported the tools I used at work to Linux by making a simple change to the Makefile. I ran an analysis on the code that takes about one month to complete on a SPARC 10 server. It took about a month on my 486 PC running Linux (try that with the popular OS).

Why you may ask, is such a robust OS not as popular as the other OS running on PC's?

If consistency is maintained, it will be. I have quietly been using Linux and watching its growth. The only real threat to the growth of Linux is if the distributors want to play the games the hardware vendors and OS vendors play, by trying to make sure software developed on their system will not run on any other system (a.k.a. locking the client in). As a developer, I continually run my code on at least two platforms to make sure it is portable. I usually try for at least three. OK so portability is almost as important as consistency, but portability requires consistency and standardization.

I have run at least three different distributions of Linux on my computers. Currently, I am running the same distribution on all three, including my laptop. I purchase a Linux distribution on CD instead of getting it off the internet so I can get the value added. In my case the biggest value added is the support (after all, I am a designer/developer, not a system administrator).

Recently, I have read rumblings about distributors thinking about providing software that will not run on other distributions. To do so would be a fatal mistake for it could do something that the popular OS vendor can not do, by dividing the Linux community. I wonder if that is the reason a major hardware vendor bought into a Linux distributor that provides multi-platform support.
—Robert Witkop
rmwitkop@cts.com

Archive Index Issue Table of Contents

Advanced search

Software Libre and Commercial Viability

Alessandro Rubini

Issue #58, February 1999

Mr. Rubini gives us his opinion of the Open Source movement.

Fortunately, Linus' project of world domination is going to come true fairly soon. The trend toward this goal can be verified by checking how the press is behaving towards GNU/Linux solutions, looking at how several educational entities are going to introduce free software in the schools and verifying Linux's usual technical excellence.

Today in 1998 (yes, it is still 1998 as I write), the most important job remaining, in my opinion, is propagating the social and commercial implications of free software. While I greatly appreciated Russell Nelson's article “Open Source Software Model” in the July issue of *LJ*, I feel the need to expand on the points he briefly touched.

Please note that I'm not an expert in economics or politics. I'm just a build-it-yourself kind of technical guy who is reporting his own experience in the battle for survival, in the hopes of helping someone else adapt to new environmental conditions. Some of these ideas have already been discussed with friends or on the Free Software Business mailing list (fsb-subscribe@crynwr.com), which I joined after reading Russell's article.

Viability for Individual Consultants

The best feature of any computer system is flexibility-- allowing users to tailor its behaviour to their own needs. This flexibility is often completely unknown to the general computer user, because proprietary software solutions tend to hide functionality behind a rigid external interface which denies any divergence from the expected behaviour—a *user's* behaviour.

When adopting free software, users are able to discover the real power of computer systems. Today I talked with a commercial consultant who never

thought that programs could be adapted to one's needs. He confessed his company has always acted the other way around—they adapted their needs to the software they use. Most users are victims of their software and don't even realize it.

Educating the user base about the extendibility of software will open new markets to independent consultants, creating new employment opportunities. Every user has different needs and solving these needs often means calling for technical support from people who tailor or enhance the relevant software. While this is not even imaginable with proprietary programs, source availability allows any problem that might arise to be quickly solved and new features to be easily added. While you may think this would quickly lead to a *perfect* software package, individual needs are so diverse and specialized that the *perfect* package will never exist.

For example, I and others wrote a program for a local physiology center to analyze data for a typical kind of experiment. During two years of use, the physicians found so many ways to enhance the program that it is now reported as better than the commercial solutions. The total of all fees they paid during these years reveals the program to be more expensive in the end than some of the commercial alternatives. This fact is not relevant to my clients, as they have exactly what they want and they know they can have more should the need arise. The program is obviously GPL and other centers expressed interest in getting a copy.

As more and more people are choosing free software to address their needs, I'm sure some software companies will try to demonize Linux and the open-source movement because they are losing market share. Such companies will probably try to demonstrate that IT employment is decreasing and that humankind is being damaged by the general adoption of free software. This whole argument is bogus; computers exist to be programmed, and the more you allow programming them, the more you build employment opportunities. If you count the number of people who offer free software consulting, you will greatly exceed any shrinkage of proprietary companies. Sticking to my previous example, the physiology lab hired my company to write the program, and other centers interested in the product are willing to hire a local consultant for installing, maintaining and enhancing our package. Did I say “enhance”? Isn't the program working? Yes, the program is working well, but there *is* room for enhancement of the product. The local lab decided to stop development “because we must run our experiment rather than invent new software features”. As anyone knows, every program has a bug and a missing feature, and this is where we build our credibility—bugs *can* be fixed and features *can* be implemented. As I suggested before, the more you make things programmable, the more they will be programmed.

Why should there be more employment opportunities in IT than there are now? First of all, because free software users have more requests for new features than users of proprietary products do, as explained above. Next, because anyone can build her own professionalism without paying to access the sources of information. I built my Linux expertise by studying source code and trying things out on my own low-end PC. Now I am confident I can solve any problem my clients might have, and my clients know I can (provided I am given enough time to deal with the problem).

Another critical point in addition to source availability is standardization on file formats, a field where proprietary products are revealing their worst features. Let's imagine an environment where every file format in the system was known: you could, for example, create indexes from any document that is produced, thus easing later retrieval. This can be accomplished off-line without any load on non-technical personnel.

Asynchronous reuse of data is “rocket science” for many users, because they are accustomed to programs that use proprietary file formats (and operating systems with no real multi-tasking or **cron** capabilities). As soon as free standards are adopted, users begin asking for customizations and are willing to pay for anything that will increase their productivity. Moreover, free standards guarantee that customers are not making the wrong bet, as they won't ever be stuck with unusable data if the software market changes.

While the conventional model of software distribution concentrates all knowledge in a few companies (or one of them), open standards leverage technical knowledge to anyone willing to learn. Whereas a proprietary product can be supported only by a limited number of qualified consultants (whose number and quality is centrally managed), the number of consultants supporting a free software solution is virtually unlimited and the offer can quickly adapt to the request.

In a world where computers are just tools to accomplish some other goals, easy customization and quick maintenance are basic requirements of power users. In my opinion, free software will quickly gain the trust it needs to be a real market phenomenon. As soon as you start to trust some open-source products, you learn that they deserve more. GNU/Linux fans must be ready to offer support in order to fulfill the upcoming need for consultants.

Viability for Support Companies

Obviously, independent consultants don't cover all the needs of computer users. Several activities can't be handled by individuals. Red Hat and S.u.S.E. are demonstrating that creating and maintaining a distribution can be a good source of revenue even when the product is freely redistributable. Debian-

based efforts are on the way, although less advanced—mainly because both Red Hat and S.u.S.E. bundled proprietary products with Linux in order to survive while the market share was low, while Debian is completely detached from proprietary products.

In addition to “creating and packaging” jobs, open-source companies can specialize in technical support, covering the situations where computer systems are of critical importance. Big business realities using computer systems in their productive environment won't be satisfied with either the external consultant or the in-house technician. They need to rely on an external structure that guarantees round-the-clock operation of their technological aids.

Even if GNU/Linux or any other operating system is demonstrated to be completely reliable, power users will need to rely on a support company as a form of insurance. The more important computers are for a production environment, the more people are willing to pay to be reassured that everything will go on working and to have someone “responsible” to call in case of any failure. Such a “power user” support contract could also include a provision for refunds in case of down time. Big support companies will be able to efficiently deal with it, and clients will be happy to pay high rates if they never need to call for assistance.

In short, I see no need for software companies to sell any product; the support environment is big enough to offer good business positions in Information Technologies. Those at the top could use some of the revenue to pay for free software development, thus gaining access to the best software before anyone else and associating their name with software products. As a matter of fact, this practice is already pursued by the big distributions.

Viability for Education Centers

Needless to say, schools and universities have the best interest in teaching information technologies using free software tools. Due to its technical superiority, free software environments have more to offer to the students, but also need more technical knowledge to be proficiently administered. I see no money saved here in choosing free operating systems over proprietary ones, but educational entities could better spend their money on hiring system administrators than on subsidizing some already-too-big commercial software company. While my country, Italy, is stuck with a few rules that offer more support for buying things rather than for increasing human resources, other countries are already moving in the right direction—Mexico and France, for example, have announced plans to use Linux in their public schools.

One more point leads toward free software in education: when students get jobs, they prefer to use tools they learned at school in order to minimize extra

learning efforts. This fact should lead colleges to teach only those tools not owned by anyone—those that are libre. Schools should teach proprietary software only if two conditions apply: no viable alternative is available, and the company that distributes such software *pays* the school for teaching its product. Paying someone for a product and then freely advertising it for him is definitely nonsense.

Social Issues

A few social issues relate to choosing one software model over another one. Although I mark them as social, they have economic implications as well.

While free software is not cheaper than proprietary software if you bill for your own time, some environments use different rates in converting time to money. Most emerging countries have good intellectual resources but little money, and they usually have many not-so-new computers as well. Proprietary operating systems are unaffordable for them, but free solutions are viable and productive. Actually, the “Halloween” document from Microsoft underlines that Linux is growing very fast in the Far East. Charity organizations usually have this same environment—little money and a good amount of human resources. This leads straight to the free software model for any IT requirement.

These ideas will probably suggest that free availability of information looks fairly leftist in spirit, as “information to the masses” looks quite similar to the old adage “power to the masses”. What is usually ignored is the strong rightist flavour of the Open Source movement. The free software arena is fiercely meritocratic and a perfect environment for free competition, where the laws of the market ensure that only the best ideas and the best players survive. Proprietary standards, on the other hand, tend to diminish competition by decreasing innovation and consolidating previous results.

Limits of the Free Software Model

Naturally, I'm aware that not every software package can easily be turned into free software. I'm not talking about office products—I'm confident some good projects will supply this need, sooner or later. Rather, I'm talking about all environments where a strong competition exists for a product only loosely based on its software component. For example, industrial equipment might include a computer and some commodity hardware (a robot, custom I/O peripherals, PLCs, etc.); the software application hosted in the computer is a minor part of the whole, but its features greatly affect the overall value of the equipment. Producing and debugging such applications usually require huge investments (preventing free redistribution of the source code), as a form of protection against competitors.

Another meaningful example is cell telephones. They include a lot of software and such software is the component that defines the overall capabilities of the device. However, this software is almost invisible to the end user, who perceives the device as a telephone and not a computer. Such software is strictly proprietary because of its major functional role in the device.

Unfortunately, I see no easy way to liberalize this type of code. Although I don't care too much about cell phones (I don't use them), I would prefer to see free industrial applications because their technological content is usually worth reusing and adapting to new problems.

Alessandro lives in one of the least Linux-aware towns in the least Linux-aware country in the world. He writes free software for a living and advocates free software for a mission. He hopes his upcoming child will keep off computers, recalling the good old times when such beasts where confined to their technical zoos. He reads e-mail as rubini@prosa.it, deleting spam and replying to everyone else.

Archive Index Issue Table of Contents

Advanced search

Announcements by Sun and Troll Tech

Marjorie Richardson

Issue #58, February 1999

UltraSPARC, Java licensing and free Qt.

On December 8, Sun Microsystems made two announcements of interest to the Linux community. One was the completion of the Linux port to the UltraSPARC architecture; the other was the new, more open licensing of Java.

UltraSPARC

When Sun joined Linux International back in May, it was with the expressed intention of joining the Linux community to do the UltraSPARC port. This has now become a reality. In addition, they have announced their intention to allow vendors to sell the UltraSPARC preloaded with Linux as well as Solaris.

Every machine sold preloaded with Linux is another win for Linux. An even bigger win is having yet one more of the “big guys” acknowledge that computers with Linux pre-installed are more attractive to potential buyers, especially those new to Linux. I for one am happy to see Sun following in the footsteps of Corel Computer and Cobalt Networks in making this decision.

Java Licensing

The new, open licensing for Java has been speculated about for some time. Will Sun make it open? If so, when? Well, they did it with this announcement-another big win for the Open Source movement. Source code has always been free for non-commercial use and the binaries have been freely available for use in tools developed by others. Here's how it has become more open, according to the press release:

- Allows commercial entities to use and modify the source code for commercial software product development without charge.

- Allows innovation of the source code without requiring that innovation be returned to Sun.
- Allows commercial entities to modify and share compatible source code with other commercial entities without charge and without mediation from Sun.
- Allows licensees to package for resale Sun's Java platform class libraries with virtual machines from other licensees.

These are major changes, but not quite the GPL. Developers who actually incorporate the code into a commercial product will still be required to pay a fee to Sun. Still, it's a step in the right direction and others are sure to follow suit.

Troll Tech

In a similar vein, Troll Tech announced in November that it plans to release version 2.0 of the Free Edition of the Qt graphical interface under an Open Source license. This will eliminate any worries and controversy regarding inclusion of the KDE desktop in commercial products.

Good news, indeed, to everyone who has wished for a user-friendly desktop for Linux. KDE has come a long way toward providing that option for those who are shy of the command line. (See “KDE: The Highway Ahead” by Kalle Dalheimer in this issue.)

Archive Index Issue Table of Contents

Advanced search

Best of Technical Support

Various

Issue #58, February 1999

Our experts answer your technical questions.

Pre-installed Slackware

I picked up a used laptop that has Slackware partitioned on the hard drive. Is there any way to log in to it without the password? I have a UNIX book but it does not address this problem. Do I have to re-install? I have no idea who the previous owner was. —Mike, shbecke@ibm.net

Try to boot the computer in single user mode (type linux single at the LILO prompt). If this does not work, you should get boot floppies in order to start a minimum LINUX in a RAM disk. Such floppies should be available in a Slackware distribution. Then you could mount the root partition by typing:

```
mount -t ext2 /dev/hda1 /mnt
```

and edit /mnt/etc/passwd in order to suppress the root password. —Pierre Ficheux, pficheux@com1.fr

One of the simplest ways to do this is to obtain a boot and root disk pair from the Slackware distribution. Boot using both and use mount(8) to mount the root file system from your laptop. You can then edit the mount_dir/etc/passwd (or mount_dir/etc/shadow) file to remove the password.For system administrators concerned with this potential security problem, the only way to prevent it is through the use of BIOS features that prevent access to the floppy drive without a password. This is not a Linux security flaw—any UNIX platform that may be booted from a set of floppies or from a CD-ROM that allows access to the hard drive has the potential to be “hacked” in this way. Notable exceptions are systems that use encrypted file systems, but those are rare and often much slower during normal operation. —Chad Robinson, chadr@brt.com

Software in RPMs

I have Slackware Linux. Wherever I look, I find software that is in RPMs. Is there no option other than shifting to Red Hat? Any conversion utilities? —Aseem Asthana, asthana@bom4.vsnl.net.in

Try the package www.chez.com/imil/stuff/rpm4everyone.tar.gz; I use it every day on an old Slackware 3.0 distribution. —Pierre Ficheux, pficheux@com1.fr

Try **alien**, a package that converts packages from one format to another. Otherwise, compile **rpm** on your system so you can do the following:

```
rpm2cpio package.rpm | cpio -list
rpm2cpio package.rpm | cpio -make-dirs -extract
```

It is quite handy. Note, however, that you might find incompatibilities between your system and the Red Hat packages. I always prefer compiling programs from source when I am on Slackware (but I might be overly cautious). —Alessandro Rubini, rubini@prosa.it

a.out

I just installed Red Hat's Linux 5.0. I used the C compiler to compile a simple C program, test.c, using the command:

```
gcc test.c
```

The program compiled and produced an executable called a.out. When I try to run a.out by typing:

```
a.out
```

I get a message that says a.out is not a recognized command. What am I doing wrong? —Jeff Miller, jeff_miller@msn.com

First confirm that the a.out file has the correct permissions. Use **ls -al a.out** to confirm that the executable (x) bit is set. If it isn't, use the chmod +x command to set this flag. If the permissions are correct, specify the full path to the file, as the current directory usually isn't in your default path. Use **./a.out** to ensure you are attempting to execute the correct program. Change your path by editing the /etc/profile file or the profile file in your home directory. —Vince Waldon, vince.waldon@cnpl.enbridge.com

Disk Space

How do I find out how much free space is left on my disk? —Kirk, sci@wadi-petro.com

Use the **df** command. Briefly, running df without arguments will show the free space (in KB) on all your disks; **df /some_directory** will show the space left on the disk that **/some_directory** is on. Also note that df has an undocumented **-h** option that shows sizes in GB or MB as appropriate—convenient for today's large disks. See the man pages for information on other options. —Scott Maxwell, s-max@pacbell.net

Relaying E-mail

The problem is the following. Each time I try to send an e-mail using a program that manages pop3 accounts such as Eudora or Netscape Mail, I receive the following message: "The recipient user@domain.com is not acceptable to your SMTP server. The message is not sendable until the recipient has been changed."

This problem appeared after we upgraded from the previous version of Linux to version 5.0. No problem occurs when receiving e-mail using these programs or when sending e-mail through Pine—only when using Eudora or any other similar program. How can we solve this? —Ricardo A. Williams L., rick@corotu.stri.si.edu

The problem is in the new security policies of Red Hat 5. Your mail gets refused with a message of "551 we do not relay". The solution here is authorizing your client machine to relay mail through the Linux server. Your /etc/sendmail.cf is quite clear about the options:

```
# file containing IP numbers of machines which can
# use our relay
F{LocalIP} /etc/mail/ip_allow
# file containing names of machines which can
# use our relay
F{LocalNames} /etc/mail/name_allow
# file containing names we relay to
F{RelayTo} /etc/mail/relay_allow
```

Add lines to the proper file describing either the client's IP, the client's name or the recipient's name. —Alessandro Rubini, rubini@prosa.it

Digital UNIX Gateway?

I currently have a TCP/IP (via modem) connection between my Linux box at home and my office workstation (DEC station running Digital UNIX). The problem is that the DEC machine is not a gateway, so I cannot reach the rest of the subnet or the rest of the world, for that matter. Is there a way my Linux box can reach the subnet gateway which is two hops away? The **route** command in my current version of Linux (Slackware, kernel 2.0.0) does not support the **-hobcount** flag, which is supported by Digital UNIX and would do the trick. —Martin Olivera, martin@pantano.ucsd.edu

If your PPP IP address is one from the subnet your DEC station is sitting on, you just need to make sure it does ARP proxying for your Linux machine (in other words, it has to accept packets for your Linux machine's IP on its local Ethernet). If this is not the case, then it is more difficult. The options you have are:

- Find out if Digital UNIX can do IP masquerading like Linux can.
- Configure **routed** on your DEC server and advertise a route pointing to your Linux machine. Note that this will not work if your default gateway ignores RIP information, and it may upset your network administrator and/or be against company policies.

—Marc Merlin, marc@merlins.org

Typing Spanish Characters

As a Spanish speaker, I want to use a keyboard with a complete set of Latin characters. I succeeded in implementing it for almost all applications, except Netscape. I installed the XKeysymDB file in the correct place, and this file works properly for Sun machines (I tested), but not for PCs with Linux. I tried to find the answer at Netscape's home page, but I couldn't. Perhaps I did the wrong search. Does anyone know how to set Netscape in order to have a “compose” key which produces accents, tildes and all that sort of thing? —Guigue, guigue@nucate.unicamp.br

If I'm not mistaken, the XKeysymDB file works only for a particular keyboard, so the one that works for your Sun keyboard is unlikely to work for your Linux machine's keyboard. Jamie Zawinski's **xkeycaps** found at http://www.jwz.org/xkeycaps/ may help; it is a graphical editor for editing keyboard setups under X. —Scott Maxwell, s-max@pacbell.net

Migrating from Windows

I got interested in Linux through a programme on BBC World. However, after an afternoon roaming the Internet, I couldn't get an answer to two questions most laymen probably have on the subject: does Linux replace Windows on my computer and is the process irreversible; will I be able to use my Windows-based programmes on Linux? —Philippe Humblé, humble@mbox1.ufsc.br

Linux can replace Windows on your computer, or the two can coexist. If you buy a commercial distribution or a Linux book, it should help explain how to do this. The process is irreversible only in the sense that you'll never want to go back. There are different ways to use your Windows-based software under Linux. A couple of emulators—Wine and Wabi—enable you to run some Windows software directly under Linux. (Similarly, DOSEMU lets you run DOS

software under Linux.) Other Windows software can be run by rebooting the machine into Windows, using the software, and then rebooting into Linux again as soon as possible. As time goes on, you'll discover Linux software that can partially or entirely eliminate your need for Windows—Linux-native word processors, spreadsheets and such. —Scott Maxwell, s-max@pacbell.net

Alternatives to X

I am new to Linux and have heard much about the X Window System when word processing using Applixware Office or Corel's Word Perfect for Linux. What if I don't want to use X? Are there quality office applications that will run without X?<\n> —John Tam, johta@mailexcite.com

There is not, as far as I know, a non-X integrated office suite, but many of the pieces exist. For document processing, you can use TeX. Whether you'll like TeX or not depends on your needs—it is rock-solid and extremely powerful, but it is not WYSIWYG. —Scott Maxwell, s-max@pacbell.net

Most people associate “quality office applications” with “graphical user interface”, a “what you see is what you get” environment like the current flock of office applications shipping for Windows machines. On UNIX systems, the standard GUI environment is the X Window System, so all of the current office suites for Linux run on top of X. This makes life much easier for the programmer—she can concentrate on writing a good word processor and leave the details of making it “graphical” to X. Early versions of X were somewhat tricky to set up and supported only a small subset of available graphics cards, but the configuration programs have come a long way as has driver support—a quick read of the X HOWTO should leave you with nothing to fear from “getting graphical” on your Linux machine. —Vince Waldon, vince.waldon@cnpl.enbridge.com

Archive Index Issue Table of Contents

Advanced search

New Products

Ellen Dahl

Issue #58, February 1999

iHTML Merchant 2.0, Auto-Changer, JetStor RAID and more.

iHTML Merchant 2.0

Inline Internet Systems, Inc. announced the availability of the iHTML Merchant 2.0 software solution for electronic commerce on the Internet. iHTML Merchant 2.0 enables business owners, web developers and ISPs to deploy sophisticated on-line storefronts. Upgrades of the software from 1.0 are $149US. Version 2.0 is $739US per web server, which includes iHTML Pro 2.16 (available separately at $590US). Contact: Inline Internet Systems, Inc., 7305 Rapistan Court, Mississauga, ON L5N 5Z4, Canada, Phone: 905-813-8800, Fax: 905-813-8286, E-mail: info@inline.net, URL: http://www.ihtmlmerchant.com/.

Auto-Changer

UniTrends Software has released Auto-Changer, tape library software for Linux. Auto-Changer allows Linux users to back up and work with tape libraries. CTAR sells for $195 on Linux (Intel) and Auto-Changer is $149 for Linux, but must be used with CTAR, the Business Critical Backup Solution.

Contact: UniTrends Software Corporation, 1601 Oak St., Suite 201, Myrtle Beach, SC 29577, Phone: 800-648-2827 (or 843-626-2878 outside the US), Fax: 843-626-5202, E-mail: sales@unitrends.com, URL: http://www.unitrends.com.

JetStor RAID

AC&NC announced the JetStor RAID desktop-sized SCSI disk array system which includes a new high-performance RAID controller as well as functional improvements. Complete information including technical specifications can be found on Advanced Computer and Network Corporation's web site. Sample pricing: 32GB for $6,500, 63GB for $7,999 and 126GB for $12,250.

Contact: AC&NC, 5001 Baum Blvd., Suite 770, Pittsburgh, PA 15213, Phone: 412-683-9010, Fax: 412-683-9070, E-mail info@acnc.com, URL: http://www.acnc.com/product_jetstor.html.

VariCAD 7.0-1.0

VariCAD has released a new version of its professional CAD system for mechanical engineering—VariCAD 7.0-1.0. VariCAD 7.0-1.0 is able to communicate with rendering software and FEM software (Cosmos, NuGraf, etc.). Free trial and demo versions are available at VariCAD's web site. The prices of the VariCAD system remain unchanged—VariCAD for Linux is $299-$499.

Contact: VariCAD, 931 Greenbriar Avenue, Ottawa, ON K2C 0J8, Canada, Phone 613-723-0953, E-mail: info@varicad.com, URL: http://www.varicad.com/.

MetaCard 2.2

MetaCard Corporation announced the release of MetaCard 2.2. New features include support for new image formats including PNG and animated GIFs, the ability to resize images on all platforms, and many new properties, commands and functions to make application development faster and easier. The free MetaCard 2.2 Starter Kit is available from the MetaCard web site.

Contact: MetaCard Corporation, 4710 Shoup Pl., Boulder, CO 80303, Phone: 303-447-3936, Fax: 303-499-9855, E-mail: info@metacard.com, URL: http://www.metacard.com/.

Macsyma 421 for Linux

Macsyma Inc. announced the release of Macsyma 421 math software for Linux. Macsyma offers mathematical power in symbolic, numerical and graphical mathematics. The PC interface offers Macsyma's MathTips Advisor, which allows users to type questions in their own words and receive executable “tips”. The U.S. commercial price for Macsyma 421 for Linux workstations is $249 (or $199 without paper manuals).

Contact: Macsyma Inc., 20 Academy Street, Arlington, MA 02476, Phone: 781-646-4550, Fax: 781-646-3161, E-mail: info@macsyma.com, URL: http://www.macsyma.com/.

HP Firehunter Supports Linux

Hewlett-Packard Company announced that its HP Firehunter family of Internet service-management solutions, targeted for the Internet service provider (ISP) community, now supports Red Hat Linux. HP Firehunter is a family of measurement and monitoring solutions designed specifically to help ISPs proactively manage Internet services such as mail, news and web functions. The product family includes Firehunter/L ($1,450) for small ISPs running up to 20 Internet servers and Firehunter 1.5 for mid-sized providers with up to 60 servers. See web site for pricing.

Contact: Hewlett-Packard Company, 3000 Hanover Street, Palo Alto, CA 94304, Phone: 650-857-1501, URL: http://www.firehunter.com/.

PlanetUplink

Planet Computer's newest business solution, PlanetUplink, has been expanded to support Linux (server and client). PlanetUplink IBN (Internet Based Network) allows businesses to gain access to and share virtually any application or database simultaneously (real time) on almost any computer from their remote and multiple offices, globally, via the Internet. IBN is available in 2 versions: Internet Office Hosting (IOH) and Internet Application Hosting (IAH). IAH/IOH user costs vary from $45 to $75 per month, in addition to setup fees depending upon the application(s).

Contact: Planet Computer, 910 16th St., Suite 624, Denver, CO 80202, Phone: 303-825-1778, Fax: 303-825-1773, E-mail: planet@planet-computer.com, URL: http://planetuplink.com/.

NetTracker 3.5 and Webmaster 3.0

Sane Solutions, LLC and COAST Software Inc. announced the offer of a package combining the NetTracker 3.5 Professional web-based usage tracking software and the COAST Webmaster 3.0 web site management software. The NetTracker 3.5 product is designed to provide a simplified solution for web site traffic analysis. COAST WebMaster 3.0 is used to monitor and maintain the content of Internet and Intranet web sites. The NetTracker 3.5 Professional/COAST WebMaster 3.0 bundle is available for $750US.

Contact: Sane Solutions, LLC, 35 Belver Ave., Suite 230, North Kingstown, RI 02852, Phone: 800-407-3570, E-mail: info@sane.com, URL: http://

www.sane.com/. COAST Software Inc., 60 Queen Street, 14th Floor, Ottawa, ON K1P 5Y7, Canada, Phone: 613-567-3201, E-mail: info@coast.com, URL: http://www.coast.com.

ICE.PPN

J. River has released ICE.PPN (Portable Private Networking) to provide secure, end-to-end transmission of data between a Windows client and a corporate UNIX server, regardless of whether a firewall is in place. A full-featured 30-day trialware version of ICE.PPN can be downloaded from J.River's web site. Suggested list price for a server license with 5 clients is $1195. Reseller pricing is available.

Contact: J. River, Inc., 125 North First Street, Minneapolis, MN 55401, Phone: 612-677-8200, E-mail: info@jriver.com, URL: http://www.jriver.com/.

O2 Object Database Management System

Ardent Software, Inc. announced the availability of the O2 Object Database Management System (ODBMS) for the Linux operating system. Ardent's O2 System, an ODMG-compliant ODBMS available on Linux, gives developers access to language bindings, OQL (Object Query Language) and O2Web, an integrated set of tools for simple and rapid development of web applications.

Contact: Ardent Software, Inc., 50 Washington Street, Westboro, MA 01581-1021, Phone: 508-366-3888, Fax: 508-366-3669, E-mail: info@ardentsoftware.com, URL: http://www.ardentsoftware.com/.

Quant-X and Altera SQL Server version 2.0

The Altera SQL Server is a complete transactional data storage and retrieval system based on the relational database model and also a programmable web server. It is a multi-user, relational database with ODBC and JDBC connectivity, demanding relatively few resources. Quant-X has provided Altera with the appropriate hardware in order to port the Altera SQL Server on the above-mentioned platform. Quant-X and Altera came to an agreement regarding the bundling of Altera SQL Server Version 2.0 with Quant-X hardware. See web site for pricing.

Contact: Altera Ltd., E-mail: webmaster@altera.gr, URL: http://www.altera.gr/.

Linux Edition of Ingres II

Computer Associates International, Inc. announced the activation of its Ingres II Linux Edition Open Beta program. This program enables customers to leverage

the Linux platform in building core business applications in an *n*-tier environment, as well as preview the new version of CAI's RDBMS. The beta version of Ingres II Linux Edition can be downloaded for free from http://www.cai.com/products/betas/. When the generally available version is released, it will also be offered as a free download.

Contact: Computer Associates International, Inc., One Computer Associates Plaza, Islandia, NY 11788, Phone: 1-888-7INGRES (888-746-4737), E-mail: info@cai.com, URL: http://www.cai.com/.

Thin Client PC X Server

GraphOn announced the world's first thin-client PC X server solution, providing high performance access to the X Window System and UNIX-based applications anywhere on an organization's Intranet, the public Internet or over dial-up. Visit the web site for pricing and further information.

Contact: GraphOn Corporation, 150 Harrison Avenue, Campbell, CA 95008, Phone: 408-370-4080, Fax: 408-370-5047, E-mail: sales@graphon.com, URL: http://www.graphon.com/.

Archive Index Issue Table of Contents

Advanced search

Color Reactiveness on the Desktop

Bowie Poag

Issue #58, February 1999

Mr. Poag describes the InSight project—designing a desktop which uses color to inform the user of what is happening with his applications.

For the seven-month period spanning July 1997 to late January 1998, I was involved in an OS development project called InSight. Part of my role within the InSight development group was to study interface designs in an attempt to further understand which aspects would still be viable and useful for users for the next five to seven years. In addition to the interface, I had the opportunity to collaborate on the design of the underlying OS, since much of what we were doing on the visible aspects of the system was tied very heavily to the underlying workings of the OS. Bringing together the design of the desktop and the underlying mechanics of the OS, we hit on what we believed to be a good idea—the concept of color-reactive destop elements.

Descriptions of Color-Reactive Elements

Lamps

Figure 1

A lamp is a window element in which the color is tied directly to the operational status of the application using that window. Simply put, it is like a status LED for that particular application. As you use the program, its lamp changes color depending upon what is happening and what you'd like it to reflect—CPU usage, program status, etc.

Let's assume you have an e-mail checker which checks your mailbox every two minutes for new mail. Most of the time the lamp in the window remains blue, meaning it is just sitting around waiting for something to do. Every two minutes it turns yellow to indicate it is busy checking your mailbox for new mail. If no

mail is found, it goes back to blue. If new mail *is* found, it turns yellow or begins flashing.

Beacons

Figure 2

Beacons are like miniature lamps, meant to be used only when applications are in an *iconified* state. If you have a window open on your desktop, it has a lamp in one corner of the window. Now, when you click the iconify button, the entire application is collapsed into an icon on your desktop. If you still want to monitor that application, collapse it to a beacon instead of an icon. In that way, you will be able to see what is going on with the application without having to constantly open and close it.

For example, suppose you are downloading five different RPMs via FTP. You can collapse each one down to a beacon with a color that reflects whether or not the download is proceeding without problems. At this point, you have five little beacon icons at the bottom of your screen and you can monitor their progress by checking if they are all still glowing a nice shade of green. You could even set it up so that the color was a function of transfer speed. Bright green could indicate a fast transfer; red could indicate a slow or dead transfer.

In order to fully understand how lamps and beacons behave, keep in mind the fact that the color of the lamp (or of the beacon) can be tied to a variety of "behavior sets" such as CPU usage, process status, or specific events which may occur within specific applications. A "behavior set" dictates what actions will produce what colors. Here are a few practical examples.

Figure 3

Suppose one of the above-mentioned FTP transfers begins to stall. One of the beacons begins to glow red and stays red for several minutes. Simply pop the window back open, kill or restart the transfer. The instant you kill that process, the other four beacons begin to glow more brightly, since you have just improved the speed of the other four by freeing up some bandwidth.

Beacons make the task of babysitting multiple applications a breeze. An entire 3-D rendering package could be collapsed down to a single beacon—one that will turn green when the rendering of a scene was complete, for example. There is no longer any need to continually pop the application back open to see what is going on.

Methodology

Figure 4

Since the behavior of color-reactive elements should be consistent throughout the desktop, a centralized point of control is needed (in the form of a control panel, for example) to allow the user top-level control. From there, it would also be wise to allow the applications, if permitted by the user, to dictate their own behavior sets. Ultimately, the user must have total and complete freedom to dictate the appearance and behavior of color-reactive elements on his or her desktop.

There are two ways to go about changing the appearance of a lamp or beacon on the desktop. With InSight, the plan was simply to change the icon on the fly by loading the appropriately-colored icon in its place. The second way, which takes considerably more CPU time to accomplish, involves hue-shifting the image data within the icon +/- 180 degrees to achieve the desired color.

The first method requires a small cache of colored icons to be present and ready to be loaded. About eight different colored lamp icons (and eight beacon icons) are usually enough to handle most situations. On the FTP site (ftp://ftp.linuxjournal.com/pub/lj/listings/issue58/) is a file called 3039.zip, containing a small archive of example lamps and beacons in different colors for you to look at and experiment with. The eight colors are clear, blue, aqua, green, yellow, amber, red and purple. That's right—the raw image data has already been provided for you. These icons are, in all respects, ready-to-use.

Implementation

Figure 5

Here's how to go about constructing a control panel to handle behavior sets for color-reactive desktop elements.

The user should be presented with a list of potential “states” (like Busy, Idle, Sleeping, Error, etc.) and then be given the ability to map the color of their choice to each state.

The Color Transition Table allows the user to specify the physical behavior of the Lamp or Beacon. A whole row of “C” means simply “for this behavior, the color always remains clear”. A repeating sequence of **BRBRBRBRBRBR** would make the lamp flash rapidly between blue and red, over and over again. To slow down the rate of blinking, use a sequence like **BBBBRRRRBBBBRRRR**.

The Color Transition Table also allows the user to specify the sequence of colors it will show to indicate each specific state. If you wanted to get the user's attention, you would probably want to make the lamp or beacon flash rapidly. This can be done by alternating the sequence of colors, like drum beats in a song. To use an analogy, the lights on a police car can be thought of as color-reactive elements. When the police car is in a state called “pursuit”, its behavior is red, blue, red, blue, red, blue.

To make a lamp or a beacon flash like it is on fire, a sequence like ROYOROYOR will make it strobe from red to orange to yellow to orange, repeatedly.

The Color Transition Table allows for a tremendous amount of flexibility when dictating the precise behavior of color-reactive desktop elements. By simply changing the entries in the table, you can do everything from solid colors to wild rainbow effects just by playing with the order of colors for each state.

Listing 1

An example behavior set is shown in Listing 1. This is what the behavior set would look like if you wanted:

- Clear color for idle
- Blue for sleeping
- Violet for low CPU usage
- Red for moderate CPU usage
- Orange for heavy CPU usage
- Yellow for severe CPU usage
- Slow blinking green/clear for attention
- Fast blinking red/clear for error
- Normal blinking aqua/clear for busy

Other Examples

Using color as a function of CPU usage, a behavior set might look like this:

- Dead: clear
- Light: purple
- Moderate: blue
- Heavy: green
- Severe: yellow
- Extremely CPU-intensive: red

As a function of process state, it might be defined this way:

- Zombie: clear
- Sleeping: purple
- Idle: blue
- Running: green
- Waiting: yellow
- Segfault/dead stop: red

As a user cue, perhaps in a 3-D rendering package:

- Waiting for user input: blue
- Busy: yellow
- Rendering: green
- Error: red
- Finished: clear

As you can see, instead of using simple solid colors, lamps and beacons can be made to flash colors, such as flashing red to indicate a catastrophic failure, alert or even an incoming message.

Figure 6. Local and Remote Window Example

Behavior

A pager program or e-mail checker could be collapsed into a beacon that would turn green whenever you had a new message waiting. A packet sniffer could be made to flash red whenever suspect ICMP packets are received. An FTP client could use its lamp to indicate the various stages of connection to a host or the progress of a file transfer.

Questions and Answers

Conclusion

I propose that the GNOME desktop should not only feature this design innovation, but use it prominently in the general layout of each window as per the recommendations given here. Let's go for it! It is a simple concept to understand, simple to implement, and its function ultimately justifies its inclusion.

Update

Since I first wrote this article, GNOME Developer Eckehart Burns has developed a color-reactive Lamp/Beacon widget to the GNOME UI library which is currently part of the GNOME CVS tree. GNOME application coders now have the ability to incorporate CR into their applications at their discretion.

All listings referred to in this article are available by anonymous download in the file ftp.linuxjournal.com/pub/lj/listings/issue58/3039.tgz.

Bowie Poag (bjp@primenet.com) is a Computer Science major at the University of Arizona. Aside from school, he is currently working as the System Administrator for the Chemistry department's computer graphics facility.

Archive Index Issue Table of Contents

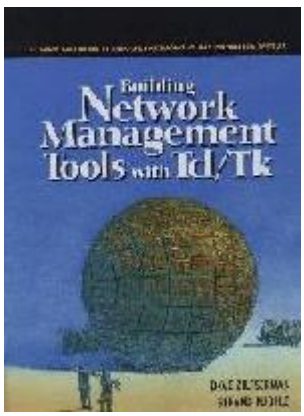
Advanced search

Book Review: Building Network Management Tools with Tcl/Tk

Syd Logan

Issue #58, February 1999

This book describes two SNMP extensions to Tcl. The first, Tickleman, is a commercial product that is available for a fee. The second, Scotty, is a package that is freely available via the Internet.



- Authors: Dave Zeltserman and Gerard Puoplo
- Publisher: Prentice Hall
- E-mail: sales@prenhall.com
- URL: http://www.prenhall.com/
- Price: $48 US
- ISBN: 0130807273
- Reviewer: Syd Logan

The stated target audience for this book is the systems administrator or network consultant who needs to develop network management software. While feature-rich third-party network management tools (for example, HP's Openview) are widely available, there may be times when an administrator or network consultant will find that the tools at his/her disposal are not flexible enough to solve a particular problem. In addition, the costs of purchasing third-

party tools can often be prohibitive; it may be impractical to spend thousands of dollars on a package, especially if one's network status monitoring needs tend to be modest.

Developing your own tools in Tcl/Tk will take some effort, but Linux comes with everything except possibly the extensions to Tcl that are needed to perform SNMP (simple network management protocol) communications. (A search on the Net indicated there is a Red Hat RPM for Scotty, one of the extensions.) The book describes two SNMP extensions to Tcl. The first, Tickleman, is a commercial product that is available for a fee. The second, Scotty, is a package that is freely available via the Internet.

For those of you who are unfamiliar with Tcl/Tk, SNMP, or RMON (remote monitor), a lot of new material is included for you to read. Those interested in learning about these technologies form a second target audience for this book, one to which I belong. I was already familiar with SNMP, having used it to develop a remote configuration tool for X terminals several years ago. I was new to Tcl/Tk, but not to scripting or GUI development. I read this book primarily to gain an understanding of Tcl/Tk. I wanted to learn about the tools, the commands and the syntax of Tcl/Tk by seeing how it could be used to solve a non-trivial programming task.

A third audience for this book is engineers interested in the design of network management software. The high-level concepts introduced should apply regardless of the programming language or GUI APIs the developer might choose to work with, although it will be up to the reader to make the transition from Tcl/Tk to whatever tool set is to be used.

Book Content

The book consists of three major sections. Section 1 includes Chapters 1 and 2, which introduce the extensions (Scotty and Tickleman) which allow Tcl to be used with SNMP. Chapter 2 provides a nice survey of the sort of network management tools the authors intend to be written using the technologies covered. Much of the information in this chapter will be new to readers unfamiliar with SNMP or TCP/IP networking and is provided with little explanation. I don't have a problem with this; the stated audience of the book, after all, is the network consultant, and my expectation is that network consultants are familiar with the network-related concepts and technologies in this chapter and elsewhere in the book. However, if you are new to technologies like SNMP, you may want to search the Web for background information or perhaps take a look at one of the books listed by the authors. They state that learning the basics of SNMP is easy. I agree, but I think the book would have benefitted from an early chapter introducing TCP/IP, SNMP and

RMON, including an overview of MIB-II variables similar to the one provided later in the book in Chapter 12.

The second section, chapters 3 through 9, introduces Tcl/Tk and further discusses the Scotty and Tickleman SNMP extensions to Tcl.

Chapter 3 is entitled “Tcl Basics”. Arrays, variables, expressions, procedures and functions, string manipulation, and flow of control aspects of Tcl are all covered in this chapter. Chapter 4, “More Tcl”, covers topics such as Tcl built-in variables, writing HTML source files, file I/O, event-driven programming and pattern matching. To illustrate pattern matching, a routine capable of validating an IP address or mask is presented and this routine is used in code later in the book. Because of this, I would suggest skimming Chapter 4, even if you are already experienced with Tcl, paying close attention to the code examples presented. If you are new to Tcl, you might want to fire up **wish** and experiment with some of the code in these chapters as you read along; this will help solidify the material as it is presented.

Chapter 5 is where all the fun begins. This chapter discusses both the Scotty and Tickleman SNMP extensions to Tcl. Reading this chapter teaches you to use SNMP to connect to an SNMP daemon on a host, and once connected, set and get MIB variables and perform other SNMP operations both synchronously and asynchronously. The chapter first deals with Tickleman and continues by showing how the same operations illustrated with Tickleman can be performed using Scotty. This chapter is important and should be studied carefully (even read twice, if necessary) before moving on.

Chapter 6 wraps up the discussion of Tcl and Scotty/Tickleman with the design and implementation of a polling loop capable of obtaining information from a number of network devices. The chapter is code intensive; I'd recommend reading slowly and carefully. It begins by describing the author's approach to maintaining data about the devices being polled (specifically, SNMP retry counts and timeout values, and community names organized by IP address and subnet). This data is stored in the form of a Tcl script at the beginning of the Tcl code that performs the polling task. When asked to poll a given device, this data is searched, first for an exact IP address match followed by a match based upon subnet mask if an exact IP match was not found. If neither of these searches results in a match, hardcoded default values are used instead. The code that performs the lookup and retrieves the data is presented in detail. The remainder of the chapter deals with implementation and use of the polling loop.

Now that the reader has a grip on Tcl and the use of Scotty and Tickleman, the authors begin a discussion of GUI development using Tk. Much as was done for

Tcl, Chapter 7 starts from the beginning, assuming no prior experience on the part of the reader. Again, I suggest firing up **wish** and trying out the Tk examples as you read. After some basic Tk widgets are introduced, the authors develop an example Tk application similar to the full-blown StatusMgr introduced in Chapter 1, making use of the Tk widgets presented earlier in the chapter. The authors do not go into any great detail when they discuss the Tk widgets introduced in this chapter; for details, you will need to refer to a Tcl/Tk reference. Their goal is to give the reader just enough information to make sense out of the Tk sample code presented. However, be aware that some of the sample code presented makes use of Tk widgets not explained in the text.

Chapter 8, “More Tk”, continues the book's coverage of Tk. The chapter starts by designing a table widget using Tk components. Next, an interactive IP path trace graphic is designed, where the code illustrates how to register and respond to mouse enter, move and leave events within a canvas. The chapter concludes with a discussion of the Itcl Mega-Widgets. These are extension widgets to Tk and are provided with the Tickleman package, or can be downloaded from the Net for use with Scotty. The authors first discuss several Mega-Widgets in a manner similar to that used to introduce Tk widgets in Chapter 7. They then use these widgets to re-implement the StatusMgr interface that was developed earlier in Chapter 7.

Chapter 9 discusses socket programming using Tcl's **socket** command. The chapter first covers some of the commands associated with Tcl sockets, then a very simple client/server example is presented. Finally, the authors describe how to build a web-based interface, so users can view network management data using a browser such as Navigator.

The third and final section of the book uses the information presented in the first two sections to build four complete network management tools. Chapter 10 develops a response time monitoring tool. This tool gathers the following statistics: current (last measured) delay, average delay, peak delay and number of completed tests as an indication of the reachability of the device being tested. The tool is capable of monitoring several network devices at once, and results are displayed in the table widget developed earlier in the book.

In Chapter 11, the authors present and discuss code that implements a network discovery tool. The code is based upon Tickleman, not Scotty. The authors are careful to point out in the source code all places that are SNMP specific; I assume this was done to identify those portions of the code that would need to be ported by readers using Scotty. It might have been helpful for the authors to supply both Scotty and Tickleman versions of the code. Chapter 11 is code-intensive; perhaps 90% of the chapter consists of annotated listings.

Chapter 12 discusses another sample application, StatusMgr. In contrast to Chapter 11, this chapter does not provide code—just a description of the program's organization and flow of control, making it a more casual read. The StatusMgr application makes use of code discussed and developed in earlier chapters, and the authors tell you where to look when needed. To get the full effect of this chapter, you can download the source code from the Internet to glance at as you read.

StatusMgr appears to be a serious network management tool, one that administrators should find useful. The application data can be accessed from a console or X terminal via the Tk user interface or from a web browser over the web. The application displays information such as device network availability, reset counts, interface uptimes, interface utilization and interface discards. It is also capable of reporting which nodes are routing the most IP traffic and are generating and receiving the most IP traffic. The application also reports historical availability information for the previous 48-hour period.

Readers should find StatusMgr a good starting point for adding their own custom network management features. The design of the user interface seems to lend itself to expansion. (Functional groups in the Tk version are organized as tab dialogs.) Since you'll have the source code and this book, you should have enough information at your disposal to tweak the code and add your own custom features.

Chapter 13 presents the next full-blown application, an IP path tracing tool. This tool is similar to **traceroute**, but with significant differences. The application not only shows the route from source to destination, but it also indicates the device types along the way (e.g., serial, Ethernet, token ring) and operational characteristics of each link (e.g., speeds, device vendor, number of packets forwarded by second). The route is depicted graphically using code developed in Chapter 8 and the operational data is displayed using the simple table widget, also developed in Chapter 8. The tool polls for operational data every 60 seconds, so it can be used to monitor the health of the link which is depicted. The authors are careful to warn the reader that the tool may not do its job 100% of the time. This is due to the use of RFC 1213 IP routing tables as opposed to the newer IP forwarding table defined in RFC 1354. Their reason for this choice is a logical one; RFC 1354 was not widely deployed at the time of the book's writing.

The chapter is well-organized, with the first section describing the tool in general. Following sections describe the code which traces the IP path, builds the user interface and polls the nodes along the path for operational status.

Chapter 14 discusses RMONv2, which can be used for application-level protocol monitoring. These tools can be used to configure RMONv2 and build an inventory of your current RMONv2 probe configurations.

Chapter 15 is the final chapter in the book. It describes how to use the Tcl plug-in for both Netscape Navigator and Microsoft's Internet Explorer. Information is provided on downloading the plug-in from the Internet. The plug-in basically allows your browser to handle HTML documents with embedded Tcl/Tk scripts. The authors present a quick overview of security policies supported by the Tcl plug-in, then go on to describe the **<EMBED>** HTML tag which is used to embed Tcl code in an HTML document. Then, the authors suggest two practical uses of Tcl plug-ins for network management. The first is to provide a graphical front-end to server-based management applications, meaning a Tcl applet embedded in an HTML page could connect via sockets to a host to retrieve network management data, which the applet would then render. The second use is to display graphs of network management report data that is stored in a server-based file or database. Examples for both of these uses of embedded Tcl are provided, along with the code necessary to implement them.

Appendix A provides a Tcl script that creates a table mapping RMON octet strings to names such as ether2.ip.tcp.kerberos. This table defines the set of protocols that an application of the type developed in this chapter is interested in configuring. The authors started Chapter 14 by developing code that connects to a device and queries RMONv2 for the protocols supported by the device. These resulting protocols are then searched for in the application-defined table. The results of this search define the protocols which can be probed by the Tcl application. Next, scripts are developed for configuring the RMONv2 Protocol Distribution Control Table, Address Map Control Table, Host Control Table and Matrix Control Table. Finally, the chapter develops code which accesses each of these tables for data. For example, code is developed which can access packet and octet counts for a given protocol (e.g., ether2.ip.tcp.nntp) using the Protocol Distribution Control Table. Not having previous exposure to RMONv2, this chapter will provide an idea of its capabilities; a look at the relevant RFCs will give a more complete overview. I'm sure this was intentional, since RMON is not the principal subject of the book. The discussion of RMONv2 Matrix Groups provides code that configures the hiMatrixControlTable , but no code to retrieve information.

Obtaining the Source Code

Instructions for obtaining the sample code and the SNMP Tcl extensions are provided in the book. The sample code is available from Net Mgmt Solutions, Inc. and requires a login and password wich are provided in the book's preface. In order to access the download page, you must give your name, phone number and e-mail address, which is a bit much as far as I am concerned.

Personally, I feel the book should have been packaged with a CD or floppy containing the latest releases of Tcl/Tk, Scotty and the sample code. The Scotty code was downloaded from the web.

Overall Impression

This book provides a comprehensive overview of the use of Tcl/Tk to develop both stand-alone and web-based network management tools. If you are a network consultant or system administrator, it should provide you with a good starting point for the development of custom tools not present in your current tool set. The authors provide information about Tcl/Tk and the Scotty and Tickleman SNMP extensions to Tcl. With this information, you should be able to start with the code from any of their sample applications and tweak it into the network management tool that best meets your needs.

For readers wanting an introduction to Tcl/Tk and SNMP, I think this book serves well to a certain degree. If you are new to SNMP or networking, you may want to look on the Internet for a more complete introduction. As far as Tcl/Tk goes, the book does a fairly good job of describing things, but eventually you will need to augment it with a Tcl reference or programming primer. The authors list (at the end of Chapter 15) additional books, newsgroups and web sites that provide information about SNMP, Tcl/Tk and network management in general.

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

LJ Interviews Informix's Janet Smith

Marjorie Richardson

Issue #58, February 1999

Janet Smith is the Director of the Linux Business Unit for Informix Software, Inc.

Janet Smith is the Director of the Linux Business Unit for Informix Software, Inc. Her responsibilities include implementing marketing strategies as well as providing solutions to product and strategic issues concerning Linux. At Informix, prior to being promoted to the Linux Business Unit, Janet held the positions of Program Manager and Senior Development Manager in Product Development. She was the Engineering Manager responsible for the first enterprise-ready database to be ported to Linux.

After graduating with a BS degree in Accounting from the University of Florida, Smith began her career with Arthur Andersen & Co. She joined Tandem Computers in 1989 and held postions in Product Management and Technical Support until 1994, when she joined Informix. Janet lives in the Bay Area with her husband, Clark, and their little boss: daughter, Veronica.

I talked to Janet by e-mail in November 1998 to discuss Linux and Informix.

Marjorie: What event first brought Linux to your attention?

Janet: It was a response from our users, rather than an event, that brought Linux to our attention. Our users, organized as the Informix International User

Group, make up an incredible collection of technical and business experts. They know the capabilities of Informix products better than anyone in the world, and we listen to their suggestions and respond to their requests.

Our users demanded the marriage of the efficient, feature-rich, “stealth” Linux operating system and the scalability, performance and extensibility of Informix database options. We delivered our first Linux products, INFORMIX-SE and INFORMIX-ESQL/C, in July at the Informix Worldwide User Conference. The response has been overwhelming.

Marjorie: What sort of evaluation procedures did you use to decide to support Linux?

Janet: User demand drove our investment in Linux. We spoke with a number of Linux users and a number of Informix users who were considering the Linux OS. We came away with a strong understanding of three things:

- The Linux market is growing very rapidly.
- The Linux market needs a robust, high-performance, feature-rich database.
- Informix has unique advantages that make it a perfect choice for the Linux community.

We saw a tremendous opportunity and we moved quickly to provide the first commercial database on Linux with our SE offering.

Marjorie: What advantages do you see in having products that support Linux?

Janet: First, it is important to understand that one of Informix's core business strategies is developing strong partnerships. We have long-standing relationships with our world-class VARs, ISVs, distributors, hardware partners and customers. Businesses rely on applications built on technology from Informix.

Linux allows our partners to move into new markets and provides a significantly lower invoice cost for delivering solutions to customers. In addition to being a “no-cost” operating system, it also provides for the excellent reuse of hardware. This opens up opportunities and allows our partners to move their solutions into new industries and markets—markets that Microsoft simply doesn't address.

The growth of the Linux OS appears to cut across nearly every vertical market, and it is gaining acceptance through every layer of the IT infrastructure.

Whether you are developing or deploying, the customer or the vendor, the VAR or the hardware vendor, everybody wins. Linux is revolutionizing our industry.

Marjorie: What are the disadvantages?

Janet: Linux has tremendous momentum right now, and the challenge is to be able to quickly respond to changes within the Linux community. There is still some standardization necessary to create a truly ubiquitous operating environment. As the first commercial vendor to deliver product on Linux, we had to work through some of those issues. Informix supports the efforts of the Linux Standards Board (LSB) and we know the Linux community is completely committed to meeting their current standards of excellence today and in the future.

Marjorie: What do you find most attractive about Linux?

Janet: The total cost of ownership is clearly one of the drivers within this market. Linux allows a much larger range of applications to be built at a much lower cost than previously imagined. Users and developers are looking for an alternative to Windows NT.

Marjorie: How do you compare Linux with other operating systems?

Janet: The acceptance of Linux is so rapid and the passion of the community so high that the same enterprise-wide tests available on other operating systems will soon benefit Linux. In general, we have found Linux to be as reliable as other UNIX platforms.

Marjorie: What other platforms do you support?

Janet: Informix products are available on UNIX, NT and Linux. Informix supports all major UNIX platforms including Sun, HP, SGI, Sequent, IBM, Compaq/DEC, Compaq/Tandem, SNI, UNIXware, Data General, Intel Solaris, NEC, Fujitsu, NCR and Hitachi.

Marjorie: Do you plan to support Linux with all your products? If not, why not?

Janet: It is clear the Linux market needs Informix solutions, and we are committed to the Linux platform. We plan on supporting Linux by providing the Informix technology that addresses the needs of the market. Our web site, as well as our Informix Developer Network, describes solutions that benefit the Linux community. We will continue to make specific announcements regarding our plans for delivering additional products on Linux.

Marjorie: Tell us about the products you have ported to Linux and why you chose to port them to Linux first.

Janet: INFORMIX-SE and INFORMIX-ESQL/C are our first entrants into Linux, because both are well-known and widely supported by our VARs and ISVs. INFORMIX-SE is an enterprise database that is extremely fast, has a small footprint, is easily embedded and requires virtually no configuration or management.

Linux developers are able to embed INFORMIX-SE as part of their applications without requiring their customers to hire a DBA to maintain and support the deployed application. It enables current INFORMIX-SE developers to deploy their applications on Linux and take advantage of Linux and the Apache Web Server. INFORMIX-SE is a fabulous fit of a high-performance database with a low-cost, low-maintenance web server.

INFORMIX-ESQL/C on Linux is for C developers building applications to access an Informix database on any platform, including Linux. INFORMIX-ESQL/C on Linux also enables developers with existing applications to easily port them to Linux. It provides the convenience of entering SQL statements directly into the C language source code. Developers can use SQL to issue commands to the Informix server and to manage the result sets of data from queries. INFORMIX-ESQL/C provides low-level control over the application for session management and error handling and gives the developer direct access to all database functions. INFORMIX-ESQL/C requires significantly less coding and is easier, faster, more productive and less error-prone than low-level direct calls to libraries.

Marjorie: Have you considered making any of your products Open Source?

Janet: Informix is committed to increasing developer participation in our products and allowing developers to add value. So, we are looking at Open Source and trying to understand which products might be appropriate within that model. It is important to remember that Informix is already the most open database available. Though not truly open source, our APIs give developers powerful ways to extend the functionality of the database. We think that meets the needs of the market today.

Marjorie: What do you think needs to be added to Linux to make it more attractive to business users?

Janet: The availability of applications is what will drive Linux in the business community. We are committed to working with our partners, VARs and ISVs to make their applications available on Linux quickly.

Marjorie: Linux seems to be climbing rapidly in popularity with all the PR it has gotten recently. How long do you expect that to last?

Janet: The growth rates posted are quite impressive, and prove Linux is not a fad. The recent PR will only fuel the fire. We think Linux is here to stay.

Marjorie: What do you see in the future for your company and Linux?

Janet: The future for Informix and Linux is not only prosperous, but also fun. We've just entered the Database Software Olympics and the Linux community bestows the medals. Technology is the Informix playing field, the games have begun, and we are going for gold.

Archive Index Issue Table of Contents

Advanced search