Linux Journal Issue #73/May 2000

Supplement

Python Supplement

Focus

Programming *by Marjorie Richardson*

Features

The Code Analyser LCLint *by David Santo Orcero*
Debugging code is never fun, but this tool makes it a bit easier.
Embedding Python in Multi-Threaded C/C++ Applications *by Ivan Pulleyn*
Python provides a clean intuitive interface to complex, threaded applications.
X/Motif Programming *by Ibrahim F. Haddad*
And God said “Let there be light”!
Parametric Modelling: Killer Apps for Linux Clusters *by David Abramson*
Get ready for parallel processing with the University of Michigan.

Forum

An Introduction to PHP3 *by John Holland*
If you are designing a new web site, this language can provide just the help you need.
Creating Smart Print Queues *by Mark Plimley*

Reviews

Columns

Python Suppliment Contents

Silly Articles

Even Sillier Articles

Not A Bit Silly Article
Book Review

Departments

From the Editor *by Marjorie Richardson*

Strictly On-Line

Python Programming for Beginners *by Jacek Artymiak*
A practical introduction to writing non-trivial applications in Python.

Archive Index Issue Table of Contents

Advanced search

Focus: Programming

Marjorie Richardson

Issue #73, May 2000

This month, we let programmers tell us how they work, from programming on clusters to building GUIs with X and Motif—it's all here. Multi-threaded programs are hot and so is Python.

Programming has become as much a part of our everyday lives as breathing. Some of us do it; some of us use it; all of us are affected by it. We encounter it when we use an ATM machine, drive a car, check out at the grocery store, and sit in front of the TV or computer. It's interesting and fun to do. Nothing beats the feeling of writing a good application that gets used by many others—it's a source of pride. Nothing feels worse than writing a good application that gets scrapped and never used. Actually, this brings up one of the good points of free software: you write it, put it out on the Web, and people who need it use it; nothing gets put in the bit bucket. Only in the commercial environment can a boss say, “We've decided to abort that project you've been working on for the last six months”--and your work disappears forever.

This month, we let programmers tell us how they work, from programming on clusters to building GUIs with X and Motif—it's all here. Multi-threaded programs are hot and so is Python. Learn how to use the two together; then use LCLint to debug all that new code you've been writing. We also have articles on Palm Pilot development tools and writing a simple plotting program (see “Strictly On-line”). We also talk again to Darryl Strauss to find out what is happening in the world of 3-D graphics.

Python

Some people collect spoons, others collect languages. Eric Raymond collects languages, and his latest find is Python. We've liked Python for a long time and so decided not only to include a feature article on Python in this issue, but also publish an entire supplement devoted to it. That supplement comes to you with this issue. We hope you enjoy it.

Contents

Marjorie Richardson, Editor in Chief

Archive Index Issue Table of Contents

Advanced search

The Code Analyser LCLint

David Santo Orcero

Issue #73, May 2000

Debugging code is never fun, but this tool makes it a bit easier.

Soon after C was born without function prototyping, it was agreed that code debugging was an excessively complex task. As a result, a very powerful tool, **lint**, was created which was able to make a large number of static verifications of the code. This interesting program was written by S.C. Johnson in the early '70s, and it was the first static code-validation tool.

With the creation of the ANSI C standard, some lint verifications became superfluous. However, lint continued to be used while other tools were being created. John Guttag and Jim Horning took a step forward by programming **LCLint**. In a rough approximation, it works the same way as lint, and lint's users will need very little time to understand how to use it. However, with a bit of work, we can improve the detail of analysis we received from lint by using LCLint. Since its birth it has received the financial support of ARPA, NSF and DEC ERP.

Today, the LClint source code is freely available. It was written in C; therefore, anyone with a standard ANSI C compiler can recompile it for their machine. Linux users can download it as a tar archive file and as an RPM package. It is available in both source and executable format. Some distributions, such as SuSE Linux 5.3, currently provide it as part of the distribution. It can be downloaded from http://linuxdoc.org/.

The original lint has far fewer commands than LCLint, and all can be emulated using LCLint. That is why we can use LCLint to replace lint. Much of the difference between lint and LCLint is that lint's annotations are related to minimizing the number of spurious messages, while LCLint's annotations have more features, only some of which are used for this purpose.

lint Commands

All lint commands are fully supported by LCLint, but we can obtain a more accurate analysis using LCLint annotations. The equivalencies between the most common annotations of lint and a more accurate version of the commands are shown in the sidebar “lint Commands”.

The basic idea of LCLint is to run it on our code before compiling. **lint** will search for many possible bugs in the program and return its findings. However, the quantity of bug candidates, though larger than the **-Wall** option of **gcc**, is still small because LCLint makes an analysis of specified code, but does not know if that code was written according to its semantics. As a result, we can use LCLint in one of several forms. The first is to use it as a simple static analyser of raw code. Although, in this mode, LCLint recognizes many mistakes without a big-time investment in the process, it is recommended only for debugging code written by other programmers. This mode of analysis is not the best, because LCLint does not know the semantics used when the code was written and therefore it cannot discover if the program does whatever it theoretically is supposed to do.

When we wish to use it on our own programs, we indicate the semantics of our program through annotations which give LCLint more information on the program functions. LCLint uses that information for deducing whether the program does what we intended and not something else. This form of using LCLint allows us to reduce the debugging effort of large amounts of code with a minimal time investment. I will discuss some annotations to demonstrate the use of LCLint. For more information, read the manual.

A third analysis method is based on the specification of abstract data types. We can specify interfaces, functions, types and predicates using the pseudo-language LCL. LCLint will automatically generate the corresponding header files to such specifications and verify that the code agrees with these specifications. However, although LCL is an algebraic-specification language, LCLint will not prove theorems or properties, verify correctness or completeness of the algebraic definition, or generate code. It will generate headings and verify that the code corresponds to the algebraic definition.

Levels of Analysis Depth in LCLint

LCLint is a versatile tool because it can do code analysis at any level of abstraction we wish. LCLint has several levels of analysis, corresponding to a greater or smaller quantity of different checking techniques. These levels are:

- **weak**, which does minimal monitoring. This level should be used only for C code without annotations. LCLint static analysis could be used as a first pass.

- **standard** is the default analysis. It does the same as **weak** analysis, plus a bit more. It needs a few annotations to be truly useful.
- **checks** does all the checks of **standard**, plus complete monitoring of the function parameters and inconsistencies. It is a good level for proving definitely that no statically verifiable mistakes are present before using other software testing techniques.
- **strict** makes absurdly strict checks. Should be used only if there is a very subtle but annoying mistake, or if we are fanatics of strong-typed and strong-structured languages. It includes checks such as if the global variables specified within a function are usable in that function (usable or not as marked by annotations), and checking of types is quite strict. The programmers of LCLint assure me that they will give a prize to anyone who writes a real program that receives no warning when using LCLint in this mode.

In addition to these generic analysis options, for each possible check we have an option to enable one and disable another.

We can gain the best use of LCLint using an incremental approach. First of all, compiling the program using **Wall** can help us find the biggest errors; for example, truly strange operations with pointers or arguments passed to functions in the wrong way.

After this, we can begin with the weak analysis. If we use a naming convention —strongly recommended—it is a good time to use the flags to test that we used the right names and used variables and macros in the right way. This is a deeper level of analysis than we can do without annotations; if we used annotations, we can get that kind of analysis.

The highest level of analysis commonly used is checks. If we obtain only spurious analysis, it is time for dynamic debugging techniques.

The strict level of analysis is good for pieces of code where we cannot find an error that we know exists.

Annotations of LCLint

All LClint annotations have a common syntax, which is

```
/*@command@*/
```

All annotations are inserted within code in the same places we can insert comments; thus, its presence in the code does not affect normal compilation.

The most frequent location of the annotations is near the location of modifying semantics; e.g., in the definition of types, if the annotations are going to affect that type.

LCLint Annotations

LCLint has almost a hundred commands. Some of the most useful ones are shown in the sidebar “LCLint Annotations”. We are not activating or deactivating types of checks with these commands, but enriching the code so that LCLint has information on the semantics of the program and is able to do its analysis more accurately.

Naming Conventions

The use of naming conventions is a programming technique which has many users, but also many detractors. LCLint does not force you to use naming conventions, but it contains support for some of them. Supported naming conventions are Slovak, Czech and Czechoslovakian.

Slovak-Naming Convention

The rule of the Slovak-naming convention is that identifiers are constructed with the scheme **abstracttypeVarname**. The abstract type and the identifier name are separated with the first character of the identifier name in upper case. The annotations of LCLint related to the Slovak naming convention are shown in the sidebar. Remember that a type's name must never have a capital letter when using the Slovak-naming convention.

Czech-Naming Convention

The rule of the Czech-naming convention is that identifiers are constructed with the scheme **abstracttype_varname**. The abstract type name and the identificator name are separated by an underline character. The modifiers related to Czech-naming convention are shown in the sidebar. Remember that a type's name must never have an underline character when using the Czech-naming convention.

Czechoslovakian-Naming Convention

The Czechoslovakian-naming convention is the same as using Czech- and Slovak-naming convention at the same time. That is why there are valid Czech and Slovak indentificators in the Czechoslovakian-naming convention. The modifiers related to Czechoslovakian naming convention are shown in the sidebar. Remember that a type's name must never have an underline character or a capital letter when using the Czechoslovakian-naming convention.

David Santo Orcero (irbis@df.ibilce.unesp.br) is working on his Dr. Sc. degree in molecular biophysics at IBILCE (Brazil). (www.biocristalografia.df.ibilce.unesp.br/irbis) He received a grant for a FAPESP research project, “Parallel computational methods for molecular biophysics”. He does scientific research on tridimensional structures of haemoglobines and snakes' toxins using a Linux cluster.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Embedding Python in Multi-Threaded C/C++ Applications

Ivan Pulleyn

Issue #73, May 2000

Python provides a clean intuitive interface to complex, threaded applications.

Developers often make use of high-level scripting languages as a way of quickly writing flexible code. Various shell scripting languages have long been used to automate processes on UNIX systems. More recently, software applications have begun to provide scripting layers that allow the user to automate common tasks or even extend the feature set. Think of all the well-known applications you use: GIMP, Emacs, Word, Photoshop, etc. It seems as though all can be scripted in some way.

In this article, I will describe how you can embed the Python language within your C applications. There are many reasons you would want to do this. For instance, you may want to provide your more advanced users with the ability to alter or customize the program. Or maybe you want to take advantage of a Python capability, rather than implement it yourself. Python is a good choice for this task because it provides a clean, intuitive C API. Since many complex applications are written using threads, I will also show you how to create a thread-safe interface to the Python interpreter.

All the examples assume you are using Python version 1.5.2, which comes pre-installed on most recent Linux distributions. The API to access the Python interpreter is the same for both C and C++. There are no special C++ constructs used, and all functions are declared extern “C”. For this reason, the concepts described and the example code given here should work equally well when using either C or C++.

Overview of the Python C/C++ API

There are two ways that C and Python code can work together within the same process. Simply put, Python code can call C code or C code can call Python code. These two methods are called “extending” and “embedding”, respectively.

When extending, you create a new Python module implemented in C/C++. This allows you to provide new functionality to the Python language that cannot be implemented in Python. For instance, several core Python modules such as “time” and “nis” are implemented as C extensions, while others are written in Python. You never notice the difference between C and Python modules, because the act of importing and using these modules is the same. If you look around in your /usr/lib/python1.5 directory, you may see some shared library files (extension .so). These are Python module extensions written in C. You will also see various Python files (extension .py) which are modules written in Python.

Typically, when you embed Python, you will develop a C/C++ application that has the ability to load and execute Python scripts. The application will be linked against the Python interpreter library, called libpython1.5.a, which provides all functionality related to evaluating Python code. There is no Python executable involved, only an API for your application to use.

Embedded Python

Listing 1

Embedding Python is a relatively straightforward process. If your goal is merely to execute vanilla Python code from within a C program, it's actually quite easy. Listing 1 is the complete source to a program that embeds the Python interpreter. This illustrates one of the simplest programs you could write making use of the Python interpreter.

Listing 1 uses three Python-specific function calls. **Py_Initialize** starts up the Python interpreter library, causing it to allocate whatever internal resources it needs. You must call this function before calling most other functions in the Python API. **PyEval_SimpleString** provides a quick, no-frills way to execute arbitrary Python code. Interpretation of the code is immediate. In the above example, for instance, the **import sys** line causes Python to import the **sys** module before returning control to the C/C++ program. Each string passed to PyEval_SimpleString must be a complete Python statement of some kind. In other words, half statements are illegal, even if they are completed with another call to PyRun_SimpleString. For example, the following code will not work properly:

```
// Python will print first error here
PyRun_SimpleString("import ");<\n>
// Python will print second error here
PyRun_SimpleString("sys\n");<\n>
```

Py_Finalize is the last Python function which any application that embeds Python must call. This function shuts down the interpreter and frees any

resources it allocated during its lifetime. You should call this when you are completely finished using the Python library. When you call Py_Finalize, Python will unload all imported modules one by one. Many modules must execute their own clean-up code when they are unloaded in order to free any global resources they may have allocated. For this reason, calling Py_Finalize can have the side effect of causing quite a bit of other code to run.

PyEval_SimpleString is just one way to execute Python code from within your C applications. In fact, there is a whole collection of similar high-level functions. **PyEval_SimpleFile** is just like PyEval_SimpleString, except it reads its input from a **FILE** pointer rather than a character buffer. See the Python documentation at www.python.org/docs/api/veryhigh.html for complete documentation on these high-level functions.

In addition to evaluating Python scripts, you can also manipulate Python objects and call Python functions directly from your C code. While this involves more complex C code than using PyEval_SimpleString, it also allows access to more detailed information. For example, you can access objects returned from Python functions or determine if an exception has been thrown.

Extending Python

When you embed Python within your application, it is often desirable to provide a small module that exposes an API related to your application so that scripts executing within the embedded interpreter have a way to call back into the application. This is done by providing your own Python module, written in C, and is exactly the same as writing normal Python modules. The only difference is your module will function properly only within the embedded interpreter.

Extending Python requires some understanding of how the Python interpreter manipulates objects from C. All function arguments and return values are pointers to PyObject structures, which are the C representation of real Python objects. You can make use of various function calls to manipulate PyObjects. Listing 2 is a simple example of a Python module extension written in C. This is the source to the Python **crypt** module, which provides one-way hashing used in password authentication.

Listing 2

All C implementations of Python-callable functions take two arguments of type PyObject. The first argument is always “self”, the object whose method is being called (similar to the infamous “this” pointer in C++). The second object contains all the arguments to the function. **PyArg_Parse** is used to extract values from a PyObject containing function arguments. You do this by passing, in the

PyObject which contains the values, a format string which represents the data types you expect to be there, and one or more pointers to data types to be filled in with values from the PyObject. In Listing 2, the function takes two strings, represented by **"(ss)"**. **PyArg_Parse** is similar to the C function **sscanf**, except it operates on a PyObject rather than a character buffer. In order to return a string value from the function, call **PyString_FromString**. This helper function takes a **char*** value and converts it into a PyObject.

Python, C and Threads

C programs can easily create new threads of execution. Under Linux, this is most commonly done using the POSIX Threads (pthreads) API and the function call **pthread_create**. For an overview of how to use pthreads, see "POSIX Thread Libraries" by Felix Garcia and Javier Fernandez at http://www.linuxjournal.com/lj-issues/issue70/3184.html in the "Strictly On-line" section of *LJ*, February 2000. In order to support multi-threading, Python uses a mutex to serialize access to its internal data structures. I will refer to this mutex as the "global interpreter lock". Before a given thread can make use of the Python C API, it must hold the global interpreter lock. This avoids race conditions that could lead to corruption of the interpreter state.

The act of locking and releasing this mutex is abstracted by the Python functions **PyEval_AcquireLock** and **PyEval_ReleaseLock**. After calling PyEval_AcquireLock, you can safely assume your thread holds the lock; all other cooperating threads are either blocked or executing code unrelated to the internals of the Python interpreter, and you may now call arbitrary Python functions. Once acquiring the lock, however, you must be certain to release it later by calling PyEval_ReleaseLock. Failure to do so will cause a thread deadlock and freeze all other Python threads.

To complicate matters further, each thread running Python maintains its own state information. This thread-specific data is stored in an object called **PyThreadState**. When calling Python API functions from C in a multi-threaded application, you must maintain your own PyThreadState objects in order to safely execute concurrent Python code.

If you are experienced in developing threaded applications, you might find the idea of a global interpreter lock rather unpleasant. Well, it's not as bad as it first appears. While Python is interpreting scripts, it periodically yields control to other threads by swapping out the current PyThreadState object and releasing the global interpreter lock. Threads previously blocked while attempting to lock the global interpreter lock will now be able to run. At some point, the original thread will regain control of the global interpreter lock and swap itself back in.

This means when you call PyEval_SimpleString, you are faced with the unavoidable side effect that other threads will have a chance to execute, even though you hold the global interpreter lock. In addition, making calls to Python modules written in C (including many of the built-in modules) opens the possibility of yielding control to other threads. For this reason, two C threads that execute computationally intensive Python scripts will indeed appear to share CPU time and run concurrently. The downside is that, due to the existence of the global interpreter lock, Python cannot fully utilize CPUs on multi-processor machines using threads.

Enabling Thread Support

Before your threaded C program is able to make use of the Python API, it must call some initialization routines. If the interpreter library is compiled with thread support enabled (as is usually the case), you have the runtime option of enabling threads or not. Do not enable runtime threading support unless you plan on using threads. If runtime support is not enabled, Python will be able to avoid the overhead associated with mutex locking its internal data structures. If you are using Python to extend a threaded application, you will need to enable thread support when you initialize the interpreter. I recommend initializing Python from within your main thread of execution, preferably during application startup, using the following two lines of code:

```
// initialize Python
Py_Initialize();
// initialize thread support
PyEval_InitThreads();
```

Both functions return void, so there are no error codes to check. You can now assume the Python interpreter is ready to execute Python code. **Py_Initialize** allocates global resources used by the interpreter library. Calling **PyEval_InitThreads** turns on the runtime thread support. This causes Python to enable its internal mutex lock mechanism, used to serialize access to critical sections of code within the interpreter. This function also has the side effect of locking the global interpreter lock. Once the function completes, you are responsible for releasing the lock. Before releasing the lock, however, you should grab a pointer to the current PyThreadState object. You will need this later in order to create new Python threads and to shut down the interpreter properly when you are finished using Python. Use the following bit of code to do this:

```
PyThreadState * mainThreadState = NULL;
// save a pointer to the main PyThreadState object
mainThreadState = PyThreadState_Get();
// release the lock
PyEval_ReleaseLock();
```

Creating a New Thread of Execution

Python requires a PyThreadState object for each thread that is executing Python code. The interpreter uses this object to manage a separate interpreter data space for each thread. In theory, this means that actions taken in one thread should not interfere with the state of another thread. For instance, if you throw an exception in one thread, the other snippets of Python code keep running as if nothing happened. You must help Python to manage per-thread data. To do this, manually create a PyThreadState object for each C thread that will execute Python code. In order to create a new PyThreadState object, you need a pre-existing PyInterpreterState object. The PyInterpreterState object holds information that is shared across all cooperating threads. When you initialized Python, it created a PyInterpreterState object and attached it to the main PyThreadState object. You can use this interpreter object to create a new PyThreadState for your own C thread. Here's some example code which does just that (ignore line wrapping):

```
// get the global lock
PyEval_AcquireLock();
// get a reference to the PyInterpreterState
PyInterpreterState * mainInterpreterState = mainThreadState->interp<\n>;
// create a thread state object for this thread
PyThreadState * myThreadState = PyThreadState_New(mainInterpreterState);
// free the lock
PyEval_ReleaseLock();
```

Executing Python Code

Now that you have created a PyThreadState object, your C thread can begin to use the Python API to execute Python scripts. You must adhere to a few simple rules when executing Python code from a C thread. First, you must hold the global interpreter lock before doing anything that alters the state of the current thread state. Second, you must load your thread-specific PyThreadState object into the interpreter before executing any Python code. Once you have satisfied these constraints, you can execute arbitrary Python code by using functions such as PyEval_SimpleString. Remember to swap out your PyThreadState object and release the global interpreter lock when done. Note the symmetry of “lock, swap, execute, swap, unlock” in the code (ignore line wrapping):

```
// grab the global interpreter lock
PyEval_AcquireLock();
// swap in my thread state
PyThreadState_Swap(myThreadState);
// execute some python code
PyEval_SimpleString("import sys\n");
PyEval_SimpleString("sys.stdout.write('Hello from a C thread!\n')\n");
// clear the thread state
PyThreadState_Swap(NULL);
// release our hold on the global interpreter
PyEval_ReleaseLock();
```

Cleaning Up a Thread

Once your C thread is no longer using the Python interpreter, you must dispose of its resources. To do this, delete your PyThreadState object. This is accomplished with the following code:

```
// grab the lock
PyEval_AcquireLock();
// swap my thread state out of the interpreter
PyThreadState_Swap(NULL);
// clear out any cruft from thread state object
PyThreadState_Clear(myThreadState);
// delete my thread state object
PyThreadState_Delete(myThreadState);
// release the lock
PyEval_ReleaseLock();
```

This thread is now effectively done using the Python API. You may safely call **pthread_exit** at this point to halt execution of the thread.

Shutting Down the Interpreter

Once your application has finished using the Python interpreter, you can shut down Python support with the following code:

```
// shut down the interpreter
PyEval_AcquireLock();
Py_Finalize();
```

Note there is no reason to release the lock, because Python has been shut down. Be certain to delete all your thread-state objects with **PyThreadState_Clear** and **PyThreadState_Delete** before calling Py_Finalize.

Conclusion

Python is a good choice for use as an embedded language. The interpreter provides support for both embedding and extending, which allows two-way communication between C application code and embedded Python scripts. In addition, the threading support facilitates integration with multi-threaded applications without compromising performance.

You can download example source code at ftp.linuxjournal.com/pub/lj/listings/issue73/3641.tgz. This includes an example implementation of a multi-threaded HTTP server with an embedded Python interpreter. In order to learn more about the implementation details, I recommend reading the Python C API documentation at http://www.python.org/docs/api/. In addition, I have found the Python interpreter code itself to be an invaluable reference.

email: ivan@torpid.com

Ivan Pulleyn can be reached via e-mail at ivan@torpid.com.

X/Motif Programming

Ibrahim F. Haddad

Issue #73, May 2000

And God said “Let there be light”!

"You can't just punch in 'let there be light' without writing the code underlying the user interface functions."

This article will introduce the basic concepts in building a graphical user interface in X and Motif. I'll go into a quick introduction to the X Window System and its programming model, then introduce Motif and illustrate some concepts with a sample program. Finally, we'll go through the basic Motif programming principles.

X History

The X Window System, or simply X, was originally developed at the Massachusetts Institute of Technology with support from Digital Equipment Corporation. X was developed for Project Athena to fulfill the project's need for a distributed, hardware-independent user interface platform.

X is a graphics system used primarily on UNIX systems. It provides an inherently client/server-oriented framework for displaying windowed graphics. It provides a way of writing device-independent graphical and windowing software that can be ported easily from one machine to another.

X provides functionality via a vast set of subroutine libraries. These may be called from a variety of high-level languages. However, they are most easily called from C programs.

Clients and Servers Concept

Since X is network-oriented, the applications developed for it do not need to run on the same system as the one supporting the display. The programmer doesn't need to worry much about the practicality of this, as X normally makes this transparent to the user.

The Server

The server is the program that controls the display. It acts as a bridge between user programs (i.e., clients or applications) and the resources of the local system. These run on either local or remote systems. The server performs the following duties:

- Allows access by multiple clients.
- Interprets network messages from clients.
- Allows two-dimensional graphics display.
- Maintains local resources such as windows, cursors, fonts and graphics.

The Client (or Application Program)

The client in X usually consists of two parts. One, the graphical user interface, which is written in one or more of Xlib, Xt or Motif. Two, the algorithmic or functional part of the application where the input is received from the interface and processing tasks are defined.

The Programming Model

In this section, I will briefly describe the main tasks of the three levels of the X/Motif programming model. The X/Motif programming model consists of Xlib, Xt Intrinsics and Motif.

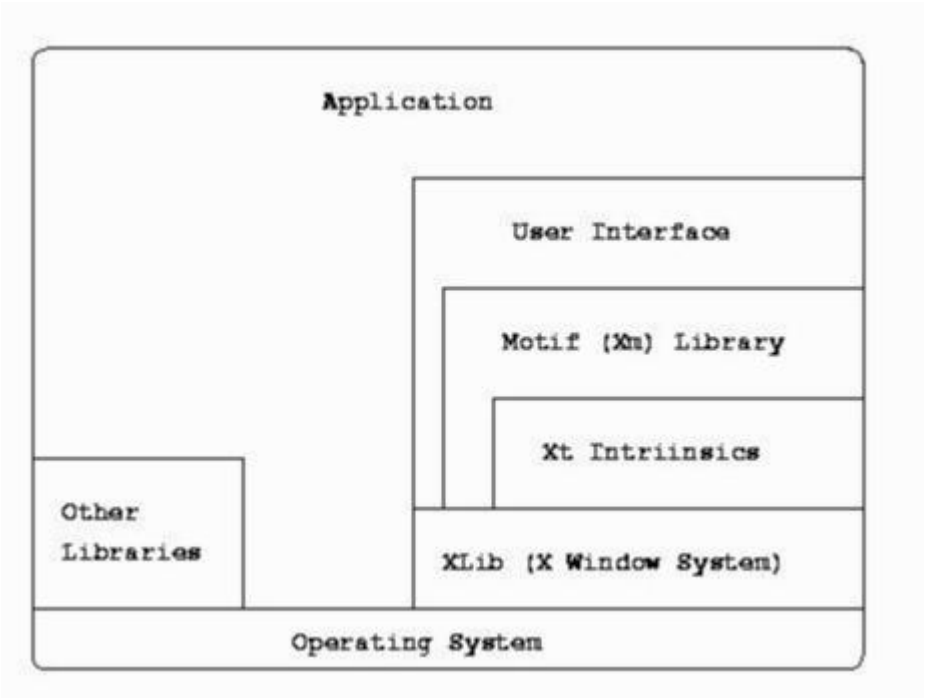


Figure 1. User Interface Library Model

Xlib

Xlib handles the interface between the client application and the network. It is part of the X software architecture. The main task of Xlib is to translate C data structures and procedures into the special form of X protocol messages, which are then sent off. The converse, receiving messages and converting them to C structures, is performed by Xlib as well.

Xt Intrinsics

X Toolkit Intrinsics (Xt Intrinsics) is a toolkit that allows programmers to create and use widgets. Toolkits, such as the Xt Intrinsics, implement a set of user interface features or application environments such as menus, buttons or scroll bars (referred to as widgets). Since Motif is built upon Xt, we'll need to call some Xt functions. However, we do not need to understand the workings of Xt, as Motif takes care of most things for us.

The client and server are connected by a communication path called the connector. This is performed by a low-level C language interface known as Xlib. It is true that many applications can be written solely using Xlib, but in general, it will be difficult and time-consuming to write complex GUI programs only in Xlib. Many higher-level subroutine libraries, called toolkits, have been developed to remedy this problem. One of these toolkits is Motif.

The Motif Toolkit

Motif is a widely accepted set of user interface guidelines developed by the Open Software Foundation and its member companies around 1989, and supported since. These rules specify how an X Window System application should “look and feel”. Motif includes the Motif Toolkit (Xm), which enforces a policy on top of the X Toolkit Intrinsics (Xt). Xt is really a “mechanism, not policy” layer, and Xm provides the specific look and feel. For example, Xt does not insist that windows have titlebars or menus, but it provides hooks for developers of specific toolkits (Motif, OpenLook, Athena widgets) to take advantage of. Motif also includes the Motif Style Guide document which details how a Motif user interface should look and behave to be “Motif compliant”.

The Relation Between Xlib and Motif

As illustrated in Figure 1, the application may interact with all layers of the windowing system, the operating system and other libraries as needed. On the other hand, the user interface portion of the application should restrict itself to the Motif, Xt and Xlib libraries whenever possible.

Motif Widgets

The widget is the basic building block for the Graphical User Interface (GUI). It is common and beneficial for most GUIs assembled in Motif to look and behave in a similar fashion. Each widget in Motif is provided by default actions. Motif also prescribes certain other actions that should, whenever possible, be adhered to. Information regarding Motif GUI design is provided in the Motif Style Guide.

Each widget is defined to be of a certain class. All widgets of that class inherit the same set of resources and callback functions. Motif also defines a whole hierarchy of widget classes. There are two broad Motif widget classes that concern us. The Primitive widget class contains actual GUI components, such as buttons and text widgets. The Manager widget class defines widgets that hold other widgets.

A Motif widget may be regarded as a general abstraction for user-interface components. Motif provides widgets for almost every common GUI component, including buttons, menus and scroll bars. Motif also provides widgets whose only function is to control the layout of other widgets, enabling fairly advanced GUIs to be easily designed and assembled.

Widgets are designed to operate independently of the application except through well-defined interactions, known as callback functions. This takes much of the mundane GUI control and maintenance away from the application programmer. Widgets know how to redraw and highlight themselves and how

to respond to certain events such as a mouse click. Some widgets go further than this; the Text widget, for example, is a fully functional text editor with built-in cut and paste as well as other common text-editing facilities.

Widgets are very useful because they simplify the X programming process and help preserve the look and feel of the application so it is easier to use.

The Motif Reference Manual provides definitions on all aspects of widget behavior and interaction. Basically, each widget is defined as a C data structure whose elements define a widget's data attributes, or resources and pointers to functions, such as callbacks.

The general behavior of each widget is defined as part of the Motif (Xm) library. In fact, Xt defines certain base classes of widgets which form a common foundation for nearly all Xt-based widget sets. Motif provides a widget set, the Xm library, which defines a complete set of widget classes for most GUI requirements on top of Xt.

"Hello World!" Motif Program

The helloworld.c program explains the steps to follow in writing Motif programs.

helloworld.c creates a window with a single pushbutton in it. The button contains the string "Hello World!". When the button is pressed, a string (Hello to you too!) is printed to standard output. This program illustrates a simple interface between the Motif GUI and the application code.

The program also runs forever! This is a key feature of event-driven processing. For now, we will have to quit our programs using the operating system:

- Use **CTRL**-c to quit from the command line.
- Use the Window Menu "quit" option.
- Depress right mouse button down around the top perimeter of the window, and choose the "quit" option from menu.

The complete program listing for the helloworld.c program is in Listing 1. The display of helloworld.c on screen will look like Figure 2.

Figure 2. helloworld.c Display

Listing 1

Compiling Motif Programs

To compile a Motif program, we have to link it with the Motif, Xt and Xlib libraries. The compile command I used for helloworld.c is

```
gcc helloworld.c -o helloworld -lXm -lXt
-lXext\
-lICE -lSM -lX11
```

Note that this compile line is required in my environment. It may be different in yours. You should check your local system documentation for the exact compilation directives.

Having successfully compiled your Motif program, the command

```
./helloworld &
```

will run it and display the PushButton on the screen as shown in Figure 2.

Function Calling Conventions

When writing a Motif program, you will be calling upon Motif and Xt functions and data structures explicitly. In order to distinguish the various toolkits, X adopts the following convention:

- Motif function and data structure names begin with Xm, such as **XmStringCreateSimple** and **XmStringFree**.
- Xt Intrinsics functions and most data structures begin with Xt, such as **XtVaAppInitialize** and **XtVaCreateManagedWidget**. The widget data structure is an exception to this rule.
- Xlib functions and most data structures begin with X. There are no Xlib functions used in helloworld.c. However, an example of an Xlib function call is **XDrawString** or **XDrawLine**.

Header Files

Any application that uses the Motif toolkit must include a header file for each widget it uses. Every Motif widget has its own header file, so we have to include the Xm/PushB.h file for the pushbutton widget, the Xm/DrawingA.h for the drawing widget and so on. However, we do not have to explicitly include the Xt header file, since Xm/Xm.h does this automatically. Every Motif program will include Xm/Xm.h, the general header for the motif library.

Basic Motif Programming Principles

We'll now analyze the helloworld.c program in detail. There are six basic steps that nearly all Motif programs have to follow. These are:

1. Initializing the toolkit
2. Widget creation
3. Managing widgets
4. Setting up events and callback functions
5. Displaying the widget hierarchy
6. Entering the main event-handling loop

Initializing the Toolkit

The first step in a Motif program is to initialize the Xt Intrinsics toolkit. Before an application creates any widget, it must initialize the toolkit. There are several ways to initialize the toolkit. The most common is XtVaAppInitialize. When the XtVaAppInitialize function is called, the following tasks are performed:

- The application is connected to the X display.
- The command line is parsed for the standard X command-line arguments.
- Resources are created using the app-default file, if any.
- The top-level window is created.

XtVaAppInitialize takes several arguments.

The Application context: the first argument to XtVaAppInitialize is the address of an application context, which is a structure that Xt uses to manage some internal data associated with an application. For the Motif program we are considering, we need not know anything about this except that we need to set it in our program.

Application class name: the second argument is a string which defines the class name of the application. It is used to reference and set resources common to the application or even to a collection of applications.

Command-line arguments: the third and fourth arguments specify a list of objects as special X command-line arguments. The third argument is the list and the fourth, the number in the list. This is advanced X use and will not be considered further in this article. Just set the third argument to **NULL** and the fourth to **0**. The fifth and sixth arguments, **&argc** and **argv**, contain the values of any command-line arguments given. These arguments may be used to receive command-line input of data in standard C fashion (e.g., file names for the program to read). Note that the command line may be used to set certain

resources in X. However, these will have been removed from the **argv** list if they have been correctly parsed and acted upon before being passed on to the remainder of the program.

Fallback resources provide security against errors in other setting mechanisms. They are ignored if resources are set by any other means (i.e., using the app-default file). A fallback resource is a **NULL**-terminated list of strings. For now, we will simply set it to **NULL** since no fallback resources have been specified.

Additional parameters: the eighth parameter is the start of a **NULL**-terminated list of resource,value pairs that are applied to the top-level widget returned by XtVaAppInitialize. For now, it's a **NULL**-terminated list since there is no resource setting.

Creating and Managing Widgets

Creating a widget is referred to as instantiating it. You ask the toolkit for an instance of a particular widget class, which can be customized by setting its resources. A widget in Motif can be created by using a specific function for creating each widget or by using convenience functions for generic widget creation and even by creating and managing widgets with a single function call to **XtVaCreateManagedWidget**.

In general, we create a widget using the function **XmCreate***widgetname*. To create a pushbutton widget, we use **XmCreatePushButton**. Similarly, to create a menu bar, we use **XmCreateMenuBar**.

Most XmCreate*widgetname* functions take four arguments:

1. The parent widget (topWidget in helloworld.c)
2. The name of the created widget, a string (“Hello World! Push me” in helloworld.c)
3. Command line/resource list (NULL in helloworld.c)
4. The number of arguments in the list

The argument list can be used to set widget resources such as the widget's initial height and width.

Once a widget is created, it must be managed. **XtManageChild** is a function that performs this task. A widget's parent manages the child's size and location, determines whether the child is visible, and may also control input to the child.

When this happens, all aspects of the widget are placed under the control of its parent. The most important aspect of this is that if a widget is left unmanaged,

it will remain invisible even when the parent is displayed. This provides a mechanism with which we can control the visibility of a widget. Note that if a parent widget is not managed, a child widget will remain invisible even if the child is managed.

However, one function actually creates and manages a widget. This function is called XtVaCreateManagedWidget, which can be used to create and manage any widget.

Principles of Event Handling

An event is defined to be any mouse action (such as clicking on a button or a menu bar option) or keyboard action such as pressing **ENTER** or any input device action. The effects of an event are numerous, and include window resizing, window repositioning and invoking functions available from the GUI.

When a widget is created, it will automatically respond to certain internal events, such as a window manager's request to change size or color and changing its appearance when pressed. This is because Xt and Motif frees the application program from the burden of having to intercept and process most of these events. However, in order to be useful to the application programmer, a widget must be easily attached to application functions. Widgets must be hooked up to application functions via callback resources.

X handles events asynchronously. It basically takes a continuous stream of events and then dispatches them to applications, which then take appropriate actions.

If you write programs in Xlib, there are many low-level functions available for handling events. Xt, however, simplifies the event-handling task, since widgets are capable of handling many events for us, such as automatic redraw and response to mouse presses.

Translation Tables

The functionality of a widget encompasses its behavior in response to user events. Each widget defines a table of events, called the translation table, to which it responds. The translation table maps each event, or sequence of events, to one or more actions. Full details of each widget's response can be found in the Motif Reference material.

Adding Callbacks

Translations and actions allow a widget class to define associations between events and widget functions. For any application program, Motif will provide

only the GUI. The main body of the application will be attached to the GUI and functions called from various events within the GUI.

To do this in Motif, we have to add our own callback functions. In helloworld.c, we have a function **pushButton** which prints to the standard output.

The function **XtAddCallback** is the most commonly used function to attach a function to a widget. XtAddCallback has four arguments:

1. The widget in which the callback is to be installed (button in helloworld.c).
2. The name of the callback resource. In our example, we set XmNactivateCallback.
3. The pointer to the function to be called.
4. Client data may get passed to the callback function. Here, we do not pass any data; it is set to **NULL**.

In addition to performing a job like highlighting the widget, each event action can also call a program function.

Declaring Callback Functions

Callback functions have the following format:

```
void functionNameCallback (Widget w, XtPointer
client_data, XmPushButtonCallbackStruct *cbs)
```

The callback function parameters are:

1. The first parameter of the function is the widget associated with the function (button in our case).
2. The second parameter is used to pass client data to the function. It is not used in our sample program.
3. The third parameter is a pointer to a structure that contains data specific to the particular widget that called the function and information on the event that triggered the call. The structure we have used is a **XmPushButtonCallbackStruct**, since we are using the PushButton Widget.

There are two final actions every Motif program must perform. First, it must tell X to display or realize the widgets. This is achieved via the **XtRealizeWidget** function; realizing the widget creates the actual window for the widget. In helloworld.c, we pass the top-level widget, **topWidget**, to the function so that all child widgets are displayed. Second, the Motif program enters the main event-handling loop; the call to **XtAppMainLoop** turns control of the application over to the X Toolkit Intrinsics. Xt handles the dispatching of events to the appropriate widgets, which in turn pass them to the application via callbacks.

Closing

This article was a mere introduction to the world of X/Motif programming. We've looked at a simple Motif program to introduce the basic concepts in building the graphical user interface. For more information, see Resources.

Naming Motif

Resources

Ibrahim Haddad (ibrahim@ieee.org) is a Ph.D. student in the computer science department at Concordia University in Montréal, Canada. Ibrahim was first introduced to Linux (0.99) and Motif at the Lebanese American University. Among his interests are e-commerce, web applications, distributed objects and helping his friends at LinuxLeb.com (Linux Lebanon).

Archive Index Issue Table of Contents

Advanced search

Parametric Modelling: Killer Apps for Linux Clusters

David Abramson

Issue #73, May 2000

Get ready for parallel processing with the University of Michigan.

After nearly 20 years of research and development, there still has not been a wide-scale uptake of parallel computing technology. However, with recent advances in PC-based hardware and networking products, it is now possible to build a parallel computer from industry-standard, commercial, off-the-shelf products. Approaches such as those used in the Beowulf project (see http://beowulf.gsfc.nasa.gov/) advocate the connection of a number of Linux-based workstations, or high-end PCs, with high-speed networking hardware and appropriate software as the basis for a viable parallel computer. These systems support parallel programming using a message-passing methodology. In spite of this, there is still a dearth of good, widespread, parallel applications.

I believe there are two main reasons for the lack of applications. First, rapid changes in hardware architecture and a lack of software standards have made it difficult to develop robust, portable and efficient software. Second, much of the effort in developing a parallel program has been focused on low-level programming issues, such as how to support message passing in a portable manner. Consequently, insufficient attention has been given to high-level parallel programming environments built around niche application domains. Thus, application developers building parallel programs have been forced to consider very low-level issues, which are far removed from their base disciplines such as physics, chemistry and commerce.

In 1994, some of my colleagues and I began a research project called Nimrod, with the goal of producing a parallel problem-solving environment for a particular niche class of application, namely parametric modelling experiments. We were motivated to do the work for this project after experiencing the frustration of trying to perform some large-scale computational-modelling experiments on a distributed computer using the available tools. We were either forced to perform the work manually, or use low-level parallel

programming tools like PVM (Parallel Virtual Machine) and batch queueing systems. Neither of these automated methods matched the type of work we wanted to perform, which at the time was modelling air pollution and exploring different control strategies. The project led to the development of a commercial tool called EnFuzion (see http://www.turbolinux.com/), which runs on a variety of UNIX platforms including Linux. It has also led to a number of very successful applications, with demonstrated returns to the researchers involved. Other types of problems can be formulated as parametric experiments, and thus can also take advantage of EnFuzion.

Parametric Modelling

Parametric modelling is concerned with computing a function value with different input parameter values. The results make it possible to explore different parameters and design options. The broad approach has been made very popular by the use of spreadsheet software, in which many different options can be rapidly evaluated and the results displayed. However, rather than simply computing a function value, we are interested in running an executable program, and thus the time required to explore a number of design scenarios may be extensive. For example, a financial model may take on the order of one hour to compute one scenario on a PC. Accordingly, to consider 100 different scenarios requires 100 hours, or over four days of computing time. This type of experiment can be feasible only if a number of computers are used to run different invocations of the program in parallel. For example, 20 machines could solve the same problem in about five hours of elapsed time.

In spite of the obvious advantage to using multiple machines to solve one large parametric experiment, almost no high-level tools are available for supporting multiple executions of the same program over distributed computers. Spreadsheet software is not designed for executing programs concurrently or utilizing distributed computers. It is possible to write a program using low-level parallel programming tools like PVM (see http://www.epm.ornl.gov/pvm/), Message Passing Interface (see http://www.mpi-forum.org/) and Remote Procedure Call (RPC), to support its execution across many machines; however, this approach has a number of disadvantages.

- First, the source must be available for the program, and it must be modified to support parallel execution. This is not always possible or desirable for complex commercial software.
- Second, all aspects of distributing the task must be built into the application. Application programmers are usually not expert in both their own discipline and parallel programming; in short, it is difficult.

- Third, unless fault tolerance is built into the application, the resulting program will not be able to handle failure in the network or in individual machines. Such fault tolerance significantly complicates the application.

Accordingly, it is not surprising that few instances of this approach are used in practice.

The alternative method is to use a remote job distribution system like Platform Computing Load Sharing Facility (see www.platform.com), Network Queueing System (see www.shef.ac.uk/uni/projects/nqs) or Portable Batch Scheduler (see pbs.mrj.com). These systems allow a user to submit jobs to a virtual computer built from many individual machines. While more general than the previous parallel-programming approach, the main disadvantage of remote queueing systems is they are targeted at running jobs, not necessarily performing a parametric study. Thus, you have to build additional software for running generated jobs based on the different parameter values and after aggregating the results. This can expose the user to a number of low-level system issues, such as the management of network queues, location and nature of the underlying machines, availability of shared file systems, the method for transferring data from one machine to another, etc.

Enter TurboLinux EnFuzion

EnFuzion belongs somewhere between a user-level tool and a software development environment. EnFuzion has been designed to make it possible to build a fault-tolerant, distributed application for a parametric-modelling experiment in a time on the order of minutes. In many cases, EnFuzion requires no programming; all you do is describe the parameters to the system and give some instruction on how to run the programs. EnFuzion manages the instantiation of your code with different parameter values, sends the files to the remote machines and retrieves them, and finally, it distributes the execution. If a program or system fails, EnFuzion automatically reruns the program on another machine.

Running an EnFuzion experiment requires three phases: preparation, generation and distribution. During preparation, you develop a descriptive file called a plan. A plan contains a description of the parameters, their types and possible values. It also contains commands for sending files to remote machines, retrieving them and running the job. EnFuzion provides a tool called the Preparator which has a wizard for generating standard plan files, as shown in Figure 1. Alternatively, if you are prepared to learn EnFuzion's simple scripting language, it is possible to build a plan using a normal text editor. The plan shown in Figure 1 is typical of a simple experiment and is quite small.

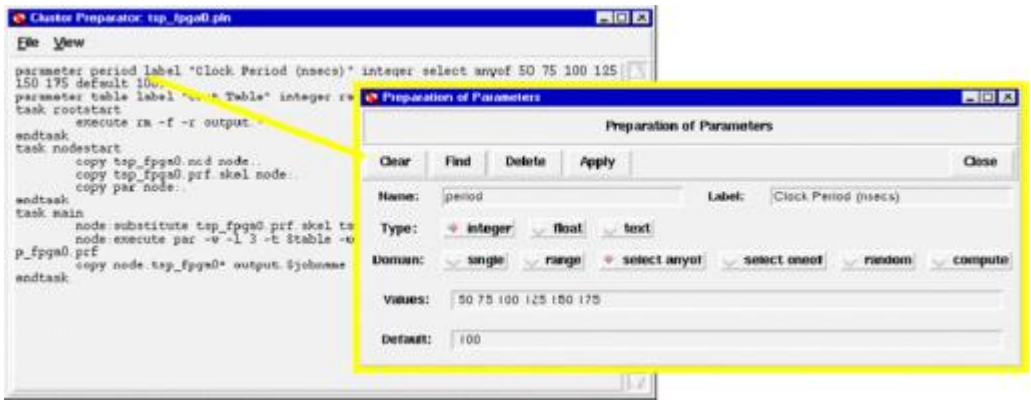


Figure 1. Plan File

The plan file is processed by a tool called the generator, which asks the user to specify the actual values for the parameters. For example, a plan file might specify that a parameter is a range of integers, without giving the start, end or increment values. The generator tool allows the user to fill in these values. It then reports how many independent program executions are required to perform the experiment by taking the cross product of all parameter values. Figure 2 shows a sample generator interaction with a CAD tool, where the clock period parameter is set to values of 75 and 100 nanoseconds and the cost table parameter is varied from 1 to 100. This interaction generated 200 individual executions of the CAD package.

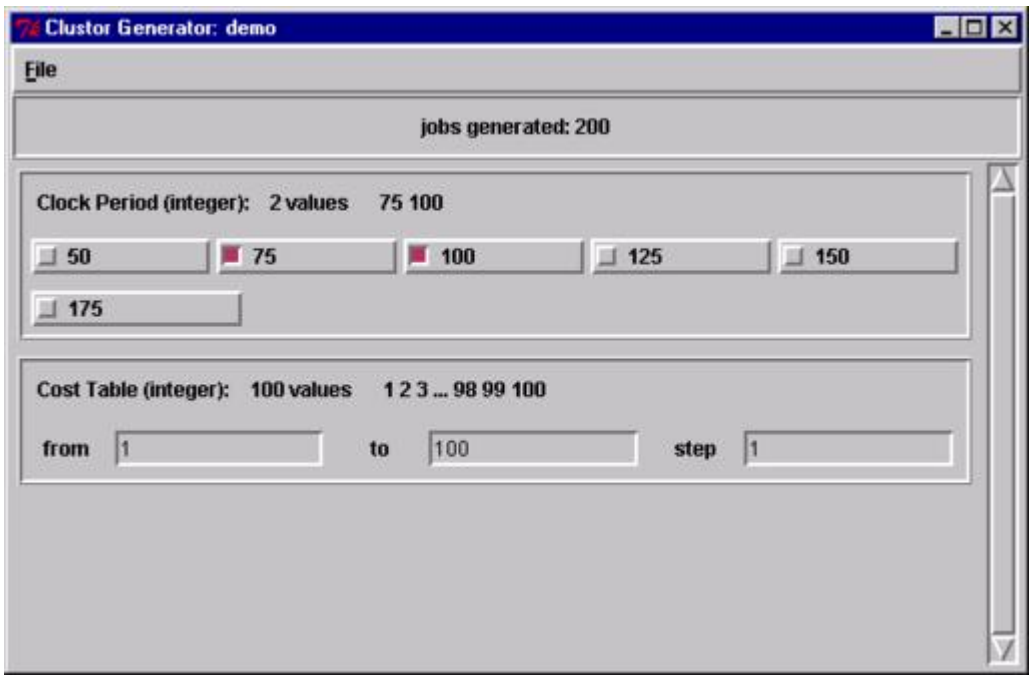


Figure 2. Sample Generator

The generator produces a run file, which contains all the information regarding what parameter values are to be used and how to run the jobs. This file is processed by a tool called the dispatcher, which organizes the actual execution. EnFuzion calls the machine on which you develop your plan the root machine. The work is performed on a number of computational nodes, as shown in

Figure 3. Files are sent from the root machine to the computational nodes as required. Output is returned to the root for postprocessing.

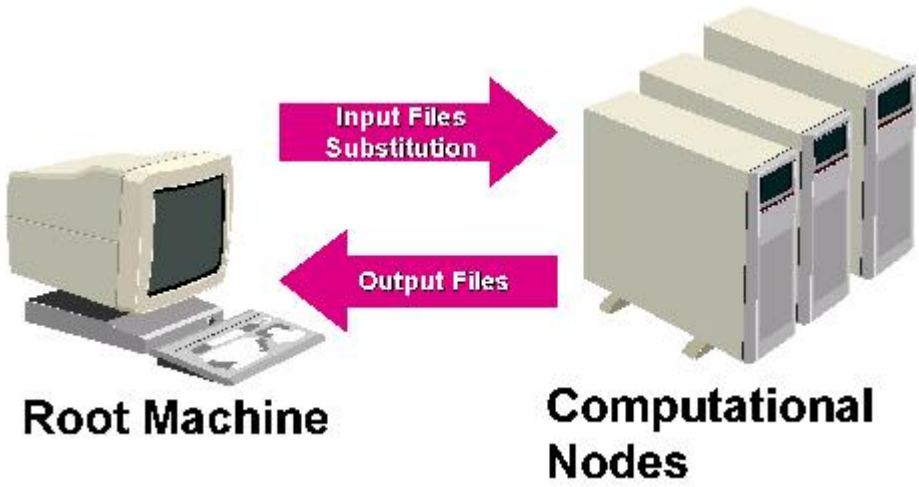


Figure 3. Computational Nodes

The dispatcher chooses to send work to machines which are named in a user-supplied file, so every user can have a different list of machines. The dispatcher contacts the nodes and determines whether it is possible to start execution of the tasks. EnFuzion allows you to restrict the number of tasks run by using many different thresholds, such as a maximum number of tasks, the hours a node will be available and the peak load average for the machine. At Monash, we have augmented EnFuzion with a simple scheme using the UNIX command **nice**. This allows a node to run more tasks than available processors, but long-running jobs are "niced" to allow short ones to have a higher priority than long-running ones. This seems to be a good way of mixing short- and long-running jobs without restricting the job mix artificially.

The Monash Parametric Modelling Engine

At Monash University, we have built a cluster of 30 dual-processor Pentium machines running Linux, called the Monash Parametric Modelling Engine (MPME). A single machine acts as the host, and is connected to both the public network and a private 100Mbit network for linking the computational nodes. Figure 4 shows part of this machine.

Figure 4. Monash Parametric Modelling Engine

Each MPME node runs a full Linux Red Hat 5.2 kernel, and the standard GNU tools, such as gcc, gdb, etc. Typically, users log in to the host and use EnFuzion to schedule work on the nodes. We have configured the system to accept up to five jobs per node, even though each node contains only two processors. To control the load on each machine, a script is run which increases the nice level of each process the longer it executes. This means it is possible to mix long- and short-running jobs on the platform. In contrast, when we limited the number of jobs to the number of processors, we found that long-running jobs were monopolizing the nodes and short-running jobs were rarely run.

To date, we have used the MPME to support a wide range of applications. Table 1 shows a list of the applications mounted during 1998 and 1999. Many of these are student projects completed as part of a semester subject. In the sidebar “A Case Study”, one of our postgraduate students, Carlo Kopp, describes use of the cluster to perform his network simulations. The results have been quite dramatic, in this case the additional computational resources allowed him to explore many more design options than he initially thought would be possible.

Table 1

A Case Study

Resources

email: davida@dgs.monash.edu.au

David Abramson (davida@dgs.monash.edu.au) is the head of the School of Computer Science and Software Engineering at Monash University in Australia. Professor Abramson has been involved in computer architecture and high performance computing research since 1979 and is currently project leader in the Co-operative Research Centre for Distributed Systems Nimrod Project. His current interests are in high performance computer systems design, software engineering tools for programming parallel and distributed supercomputers and stained glass windows.

Archive Index Issue Table of Contents

Advanced search

An Introduction to PHP3

John Holland

Issue #73, May 2000

If you are desinging a new web site, this language can provide just the help you need.

One of the latest things to emerge in web development is a language called PHP. According to its proponents at http://www.php.net/, it is in use by Volvo, Red Hat, Miss USA 1999, First USA Bank, NASA JPL and other prominent organizations. One of its most popular applications is connecting to databases, but it can also talk to LDAP, IMAP and SNMP services, generate GIFs on the fly and do other tricks.

PHP3 is a server-side include scripting language and has been compared to Microsoft ASP (Active Server Pages), but with two differences crucial to me: it is open source and it runs on Linux. Actually, it is available under a GNU license or the PHP license, which allows commercial closed-source products to be developed from it. It runs on many UNIX platforms.

What is a server-side include? Think of it as another embedded language, like JavaScript, but which is converted to HTML (or images, JavaScript or other possibilities) on the server. The user's browser doesn't see the PHP code, just the results of executing it. This protects the developer's work, enhances security and bypasses things like conflicting implementations of JavaScript.

For example, the following code:

```
<HTML><BODY><?PHP echo "Hello, Net" ?></BODY></HTML>
```

in a PHP page on a PHP-enabled server will appear in the user's browser as Hello, Net.

What can PHP do for you? It greatly reduces the complexity of using cgi-bin variables and connecting to databases, and allows programming in an easy-to-

learn language within web pages. It can do much more (the on-line reference lists 53 types of functions), but I will focus on database connectivity.

MySQL seems to be the most popular database to use with it, so I will use that for my examples. PHP can connect to Oracle, Sybase, PostgreSQL, Informix and other databases as well. It is important to note that PHP is not database-independent, unlike the DBI system used with Perl. You must use PHP functions tailored for the database you are targeting. This lets PHP access functionality available only in a specific database platform, but may reduce the portability of database-enabled projects.

In the following discussion, I will assume the reader has a knowledge of SQL, HTML and C.

I've already shown a “Hello, World” in PHP above. Note that the entire PHP code is embedded in a single tag within the HTML. The convention is to assign the extension .php3 to pages that should be pre-processed by PHP, although this is configurable. The page contains HTML (and whatever else the server and browser support, such as JavaScript) with one or more **<?php ...*php code*... ?>** blocks. As I mentioned above, the PHP is evaluated and its output sent to the browser, typically as HTML. PHP can also send any text language or even generate images on the fly.

The PHP blocks can be included anywhere in the HTML—even inside quotes or tags. For example, a link to another page could be produced from PHP. The following example:

```
<HTML><BODY>
...
For more information, visit <A HREF=
"<?php echo "moreinfo.html" ?>"> wacky link </A>
</BODY> </HTML>
```

would work, although there is little point to it. The value of PHP is that it is a powerful programming language. It allows things like links to be generated by code. One can build a whole interactive web site with it.

A Quick Overview of the Language

PHP is much like basic C, a little like Perl, and has a few unique features of its own. Here are some key facts:

- Variables are preceded by $ (just $, not @ or anything else).
- Primitive types include integers, floats, strings, arrays, associative arrays and objects.

- Type is usually set automatically, not in code, so there is no need to declare variables. They can be declared as **static** which makes them persist between PHP blocks on the same page.
- //, /*..*/ and # are all used for comments—sort of a combination of C++, Perl and shell scripting.
- Statements are terminated by semicolons.
- Functions can be declared within a block. Their declaration and call resemble C.
- Forms submitted to a PHP script pass in their variables (this is quite cool). They are accessible by name and also come in an array, which is nice when your calling form is a PHP program that may have unpredictable variable names.
- The set of control structures and operators is very similar to C, although with some extensions.
- There is limited support for object types.

Diving In

Listing 1

Listing 1 is two examples of PHP code to show some of its features. The first page accepts two numbers to add a first and last name. The passing of the form variables to be used in the next page happens automatically, which is one of the neatest things about PHP. This doesn't check their type—that they are actually numbers—in order to give an error. It does show the way PHP can be used to send and receive information to the user.

Listing 2

Listing 3

Listing 2 is an example that generates CGI variables on the fly, and the receiving page (Listing 3) deals with any incoming variables generated. If you run this in a PHP-enabled server, you'll see the variables passed in the cgi-call in the location line of the browser. All Listing 3 does is echo the names of the variables, but it demonstrates a technique that would enable a web site to generate a list on the fly and allow the user to make selections—like a shopping cart.

Listing 4

Now let's connect to a MySQL database. The following code would connect as root to a database named “stores” on the local host, with password “tiger”. It executes a query against a parts table, counts the number of rows in the query to check for the actual data, and if it exists, the result is displayed as an HTML

table with check boxes to select items. Listing 4 is the PHP code preceded by the structure of the table. Note that knowing the number of rows is handy for determining whether to display the query at all (if there is any information) as well as counting down until no data is left. Each row of data is a row in an HTML table, with the fields walked through by number. The array containing the row from the query can be addressed by number or by field name, so you could also execute:

```
$partno=$row["partno"]
```

This can be advantageous during development when field positions in the table or query may be changing more often than names.

The next page of HTML (called in the FORM command) will receive the checked-off selections as CGI variables, which can be retrieved even though their names are unknown, using the technique shown above. It will then insert into another table the items that have been checked off. See Listing 5.

Listing 5

Installation Information

The latest information on installing PHP is available at http://www.php.net/ or in the distribution itself. I have installed it for FreeBSD, Linux and Win32 using their supplied packages. I have installed it only for Apache, although it is supposed to work with other web servers. It is supplied as source code for platforms other than Win32. The installation for Apache requires recompiling Apache to add it in as a module, although if that is not an option, it can be installed as a binary and scripts can be run as cgi-bin scripts.

Conclusion

I've found PHP3 to be a very enjoyable and versatile language for web applications. As of this writing, the next version—which is primarily billed as a performance improvement—is in beta testing. (PHP 4.0 includes the ZEND engine; see their web site for details.)

Resources

John Holland programs intranet applications as a consultant for Bell Atlantic. He lives outside Washington DC with his wife and children. He can be reached at jholland@jbhsoft.linuxbox.com.

Archive Index Issue Table of Contents

Advanced search

Creating Smart Print Queues

Mark Plimley

Issue #73, May 2000

This article will help you understand print filters and how to create and install your own personalized filters.

In the December 1999 issue of *Linux Journal*, Michael Hughes introduced a few basics for setting up printing on your Linux computer. This article will go into greater detail on getting your print installation to work for a variety of printers. The biggest problem is that most applications, especially the graphics-based ones that run under X, print using the PostScript printer language, and most of us do not have a PostScript printer.

I'm basically lazy—I want my computer to make things easier for me. What's a computer for if not to automate tasks and simplify my life? It didn't take me long to become frustrated with having to deal with printing plaintext files vs. PostScript files from Netscape or other applications. So I wrote a printer filter to take care of these decisions for me, and you can, too. The BSD print system that comes with your Linux distribution has the capability to pipe print files through a filter, which can be any executable program. Since the filters usually have a simple job to do, they are normally written as a bash or Perl script.

The printer daemon sends the file to be printed to the standard input of the filter, and the filter modifies the file as needed. The filter can then write the modified file to standard output, which the print system sends on to the printer. Your filter can make any kind of modification to the file, and you don't even have to write the modified file to standard output, as we shall see later.

The BSD Printing System

Specific printer capabilities are defined in the /etc/printcap file, which is normally created when Linux is installed. Each printer or printer configuration that you use must be defined in this file. The printers can be attached locally, attached to another computer on your network or to network printers. You can

set up multiple configurations for the same printer to handle different file types, and the print system can send output to other devices or save it to a file. Virtually anything that can accept a computer file is fair game for the output device.

Each printer configuration in your printcap file defines a print queue. You print something from the command line with the **lpr** command or through a print button or menu in various applications. UNIX and Linux applications use the lpr command to make the actual print request by using the line-printer daemon, **lpd**, which spawns a copy of itself to handle each print request. **lpd** is the main program and runs continuously in the background to handle all requests from lpr, **lpq** (for querying print job and printer status) and **lprm** (to remove jobs from the print queue).

Listing 1 is a typical printcap entry that defines a printer attached to your parallel printer port. Comments in the file start with a hash (#) character. Each entry defines a printer queue and consists of a series of fields delimited with the colon (:) character. The first field of the entry supplies one or more names for the print queue, separated by the pipe (|) character. You should include “lp” as the name of one of your print queues, since it will be the default queue when you print without specifying a queue. The last sub-field of this first field should contain a description of this print queue and may be displayed by some print management software. I gave this queue an alias of “text” so that the print command could be either **lpr *filename*** or **lpr -Ptext *filename***.

Listing 1

For readability, I usually place each field on a separate line, with the backslash escape character at the end of each line except the last one. But you don't have to. It is traditional to place extra colons at the beginning and end of each field when they are given on separate lines, as in this example, with all continuation lines indented by a tab. The extra colons are ignored by lpd when it reads this file.

The fields after the first one tell lpd the properties for this queue. Each of these subsequent fields begins with a keyword, an equal (=) sign for text values or a hash (#) sign for numeric values, and the value for the field. Some keywords are switches and do not have values. In Listing 1, the second keyword, “lp”, defines the device to which the output will be sent. In this case, /dev/lp should be a symbolic link to /dev/lp0 or /dev/lp1 for your printer port. Don't confuse this field name with the default print queue name in the first field. The order of the fields does not matter, except for the first field that names the queue.

The **lf** keyword specifies the full path to the log file, where errors from the print daemon concerning this queue can be recorded. When something goes wrong, the contents of this file may give you a clue to the problem, although many print-related error messages will be sent to your system log, /var/log/messages. The **sb** boolean keyword tells lpd to print a short one-line "banner". Finally, **sd**, the spool directory keyword, lets you specify the directory path in which to place a temporary copy of the file being printed. You must specify a spool directory for each print queue. Other controlling information for the queue and each print job is also placed in this directory, so each queue in your printcap file needs to have its own spool directory.

These are the most common of the many available parameters. The printcap man page lists all the parameters that can be used to configure your print queue entries.

A Simple Print Filter

If we want to modify the print file that is sent to a queue, or perform some other task before printing, we use the output filter printcap keyword, **of**. The filter program is usually kept in the spool directory for the queue which uses it, such as /var/spool/lpd/lp1. Listing 2 shows a printcap entry that uses a simple filter. The printer port is defined by the **lp=** field as usual. When we use the **of=/var/spool/lpd/filter** field, the print daemon pipes the original file through this filter before it is sent to the device set by the **lp** field. After the **lf** field, I'm using the **sf** and **sh** boolean fields to tell the print daemon to suppress form feeds and suppress the header page for this queue.

Listing 2

So what can we do with our printer filter? Just about anything you could imagine. Listing 3 is a filter for printing plain text on HP printers. When Hewlett-Packard printers are not sent data in their native print language (such as PCL, HPGL or Postscript), they expect MS-DOS line endings consisting of an ASCII carriage return and line feed (newline). If you print a UNIX-style text file, which uses a newline only to mark the end of a line, the second and subsequent lines of your file will not start on the left edge of the paper. Without the carriage return to tell the printer to move to the left edge, you get a stairstep effect. There is no line wrap, so once the text moves off the right edge of the paper, all is lost to the proverbial bit bucket.

Listing 3

This filter uses the **awk** command to insert a carriage-return and newline at the end of each line of UNIX text. As I mentioned above, the BSD printer daemon pipes the file being printed to standard input of the filter. It then passes the

standard output of the filter to the print device specified on the **lp=** field of the printcap entry.

Getting Sophisticated

The above example works great to fix a common printing problem, but it is good for text files only. If you accidentally send a PostScript or, heaven forbid, a graphics file to this queue, you will be rewarded with lots of useless paper and a scornful system administrator.

Listing 4 is the filter we want to use to solve this problem. First, we specify a log file where we can write debugging information from the script. Next, we capture standard input to a file with the **cat** command. We can then check this file to determine what kind of file it is using the **file** command. If it is a text file, we process it as we did in our simple filter above.

Listing 4

If the file is a PostScript file, we run **ghostscript** to format it for our particular type of non-Postscript Printer. In this case, I have specified a Color Deskjet, which works for my HP Deskjet 660C. Of course, if you should be so lucky as to have a PostScript printer, you would not need this filter at all.

The last branch of our filter lets us print TIFF graphic files. We simply insert the **tiff2ps** command in front of the **ghostscript** command. In this way, you can easily add the ability to process other types of files without manually formatting them. You might want to use the full path names to the commands, since the lp daemon will not have the same path that a login shell does.

Getting Smarter

I also wanted my smart print queue to be able to deal with several different printers. You see, I use a laptop for all my e-mail and my personal computing. When I am at home, I use an old Epson-compatible printer for simple print tasks. If I need higher quality output, and my kids let me unplug it from their computer, I connect to the HP Deskjet. When I am at work, I print to an HP Laserjet 4MV PostScript printer on the network.

The change in printers is handled easily by adding information about the type of printer to the filter. To set the type of printer that is available, I added a line in $HOME/.profile to ask me which printer I will be using. This is run once when I first log in after booting up my laptop. The printer type I enter is written to the file /tmp/printer. I use the names that Ghostscript recognizes.

Listing 5 is the filter that uses the printer type information. I added the variable **PTYPE** to hold the name of the printer I am currently using. The first-level “if” branch checks for the file type, as in the previous filter. Within these branches, a second-level “if” branch checks for the different printers.

Listing 5

The other change in the filter is that instead of writing the output to standard output, I pipe it to another lpr print command. On the lpr command, I specify a print queue for the attached or network printer. This bypasses the **lp=** line in printcap. For this to work, I created another printcap entry for a print queue called “raw”. The raw printcap entry outputs to the actual printer. To create a raw print queue, just take Listing 1, remove the lp default name, and change the queue name to raw. I use a different raw print queue for the network printer, since it is a PostScript printer. Each raw printer has its own print queue in the printcap file. The raw queue for whichever printer is attached to the parallel port, and “4mvraw” is for the network printer when I'm at work.

Listing 6 is the printcap entry for the above filter and its corresponding raw print queue. One thing that is different in this entry compared to that of a normal printer queue is using /dev/null as the printer device. Since our filter will be handling the output (by printing the filtered output to the raw device), we don't need lpd to send it to any device.

Listing 6

If you have more than one printer available to you, you could use this filter technique to send any of the different types of files to different printers. And if you have a FAX modem, you could even add a “print to FAX” feature to your printing system.

Activating Your New Print Queues

After adding a new queue in /etc/printcap, you will need to tell the print daemon to reread the printcap file. Listing 7 is a script I wrote years ago to take care of the additional work necessary to activate your new printcap entries. You may want to verify the path names to the various directories and commands for your system. You must run it as superuser.

Listing 7

What if you do not always have a printer attached or available, but you wanted to send something to the print queue for printing at a later time? Will the job wait until you attach the printer or connect to the network? Yes, it will if your

raw printer is attached to your parallel port because /dev/lp waits for handshake signals from the printer.

Some Debugging Tips

If jobs fail to print, you need to determine if the problem lies with the print system or your filter. The first place to look is in your system log file, /var/log/messages, then look in the log file for the queue. Next, carefully check the syntax in your printcap file. Make sure each line except the last one for each entry ends in a "\". Turn on some debugging output to check the operation of your filter. If the problem is with the print system, you can try to stop and restart printing. You should have a script in your system startup area that does this properly. In the SuSE distribution, the startup and shutdown script is /sbin/init.d/lpd. As with any System V-style boot script, this may be manually run as /sbin/init.d/lpd start or /sbin/init.d/lpd stop.

A print queue can get stuck when the spool directory's disk gets full. This can happen for a very large job or if your disk partition is too small. You will have to remove the job with lprm to free it up, then clean up unnecessary files on that disk partition or move your spool directories to a larger partition. After copying all the directories and files to a new location, you can remove all the old directories (**rm -r lpd**) and replace the lpd directory with a symbolic link to the new location where you have copied the old lpd directory tree (**ln -s /path/to/new/location/lpd /var/spool/lpd**).

Summary

We discussed how BSD printing works and some of the common options in the printcap configuration file. We showed you how to write a filter for altering the file that is sent to the actual printer and how to write a filter that automatically processes different types of input files for printing. We also learned that a filter can redirect the output to different devices or files. Hopefully, this will give you enough information to understand and create your own "smart" printer queues and simplify your printing tasks.

> Mark Plimley (markp@blueneptune.com) started home-computing in 1978 with an Imsai 8080 (8-bit, 2MHz Intel 8080). In his former life as a mechanical engineer, he programmed in FORTRAN. He has been employed mainly as a UNIX systems administrator since 1992 and has been using Linux since January, 1995. When not in front of a computer screen, he can be found tinkering around the house, doing some activity with his wife and two teenagers or helping out at church.

All listings referred to in this article are available by anonymous download in the file ftp.linuxjournal.com/pub/lj/listings/issue73/3741.tgz.

email: markp@blueneptune.com

Archive Index Issue Table of Contents

Advanced search

Palm Pilot Development Tools

Eddie Harari

Issue #73, May 2000

How to program you hand-held computer using Linux.

One of the great things we can do with Linux and GNU tools is build an environment for cross-platform development. The perfect example for cross-platform development using Linux is the Palm Pilot development tools for Linux. In this article, I will explain the tools and methods you should use in order to program this small but very smart machine from your own Linux machine.

Background

The Palm Pilot is a very smart hand-held computer based the on the Motorola 68000 Dragon Ball CPU. A few models are on the market: PalmIII, PalmIIIx, PalmV and PalmVII. (The older machines are Palm-1000, Palm-5000, Palm-personal and Palm-professional). All of the new models use the PalmOS-3.*x* as their operating system. The basic functionality of these machines is the same. The differences between the new models are mainly the RAM size, the screen type, the battery and the shape of the machine. Details about the Palm devices can be obtained from http://www.palm.com/.

The File System

The Palm Pilot operating system uses the RAM to store and organize its “files”. Each file on the RAM is actually a Palm Pilot database. Each database has a header indicating the type and the creatorID of the database. The type can be anything: it is a 4-byte value the programmer can assign to his application or data. A type of **appl** is actually an application. If we assign a different type of database to an application, we will not be able to see it when we press the application button on the Palm Pilot. The creatorID is another 4-byte value; the PalmOS uses this value to identify the database (like a name for a file). The combination of type and creatorID is unique, so we need to apply for a

creatorID if we want to give our application to other people. Databases can store resources such as bitmaps, executable code, forms and more. An application, for example, stores its User Interface objects and its code.

The Language

To program the Palm Pilot, we first choose a programming language. Many tools and compilers can be run on our Linux desktop to program our Palm Pilot. Here, we use the **gcc** ANSI C cross-compiler to demonstrate a simple program. Palm provides its SDK (software development kit) and documentation for free, and it is downloadable from http://www.palm.com/. A handy book to have would be the MC68328 Dragon Ball manual from Motorola (for more information, see the Palm web site).

The Tools

The most important tool we need to install is the gcc cross compiler. The good news about this compiler is that by the time you read this, it should be supported by Palm itself. You should be able to find the gcc, binutils, gdb and prc-tools somewhere at the http://www.palm.com/devzone/ web site. No version of gcc comes with palmOS support out of the box yet; there are some patches to install. The patches come with the prc-tools package, containing the tools needed to combine all resources into a prc (PalmPilot resource database) file. I recommend using the gcc-2.95.2 version and not the earlier 2.7.2, because it fixes many problems. The binutils-2.9.1 should also be patched, as well as gdb-4.18, if you want to be able to debug your application. There is a good chance that by the time this is printed, this group of software will be in one big compressed tar file. The installation instructions for the cross compiler are very simple and may be found on J. Marshall's web page at http://homepages.enterprise.net/jmarshall/palmos/. (This page is about to move to the official Palm site, somewhere under the devzone section.)

Building the GUI

Palm Pilot applications use FORMS to interact with the user. These forms contain bitmaps, buttons, tables, check-boxes and many other user-interface objects. A great tool to build these forms for the Palm Pilot, called **pilrc**, was written by Wes Cherry. The current version (2.4) can be downloaded from http://www.scumby.com/scumbysoft/pilot/pilrc/. It comes as source code and has very good HTML documentation. Listing 1 is an example of a simple pilrc file which builds a simple form, alert, menu and help string. (Please note that the Bitmap with ID 10 was created directly as a resource; there is no reference to it in this file except the ID.)

Listing 1

The command

```
pilrc /usr/local/example/hw.rcp
```

will build the two UI objects in this file: the form and the string. Each of these objects will be in a separate file. We will combine these files with the actual code into a prc file (application database) later, using the **build-prc** command that comes with the prc-tools package. To view the form without actually writing the application, we can use the **pilrcui** tool. This tool is bundled with the pilrc package. For more information on the UI objects and the use of pilrc, you should look at the Palm official documentation and the pilrc HTML documentation.

The Actual Programming

Events

The PalmOS keeps a queue of events being handled by the system. These events are sent to the current application in order to execute the code to handle each event. An event is triggered when a UI button is pressed, the Palm pen touches the screen (penDownEvent) or when other operations occur. An application should transfer any events it doesn't handle to the system. When the event is handled, the PalmOS takes the next event in the queue and passes it on to the application. The application can also put events in the queue. Many PalmPilot functions cause all sorts of events to be placed in the event queue. A list of all events and what they do is in the official documentation for the Palm Pilot.

PilotMain

The main procedure of a program is called PilotMain. When the PalmOS calls PilotMain, it transfers a command to it. The PilotMain procedure should find out what the command says. The command can say, for example, that this is a normal launch of the application (**sysAppLaunchCmdNormalLaunch**) or that this is a command to find a string (**sysAppLaunchCmdfind**). PilotMain should ignore or respond to these commands in its code. An example of an application that responds to the sysAppLaunchCmdfind is the address book. Whenever we press the find button, the PalmOS sends this command to all the applications. One application that will accept this command is the address book—it starts searching for the string we entered in its database.

EventLoop

To handle all events, we should use an event loop. An event loop simply takes an event, handles it, then takes the next event in the queue. If we do not want

to handle a certain event, we can send it to the PalmOS event handlers so it will be handled for us.

Pilot API

To take advantage of all the functionality of the PalmOS, there is well-documented API for your use. You should read the API manual to see which kinds of events, functions and types of objects exist in the PalmOS. Listing 2 is simple code that writes “Hello World” on the Palm Pilot.

Listing 2

Building the Application

The first step toward running an application on the Palm is to build the resources. As mentioned earlier, we can build the resources using the command pilrc.

```
pilrc hw.rcp
```

To add the bitmap with ID 10 to the main form, I used a tool that converts ppm files to Tbmp*xxx*.bin resources. I used **xv** to convert the format of peng.gif (found at http://www.linux.org.il/) to black-and-white ppm format. Then I used **ppmtoTbmp** on the peng.ppm file, and named the result Tbmp000a.bin. The 000a in the file name is the 32-bit hexadecimal value for 10. I needed to specify this value because this is the bitmap ID I wrote in the pilrc resource file for the main form. This tool was written by Ian Goldberg, and it can be found at http://www.pilotgear.com/.

The next step is to compile the C code. To do this, we use the gcc m68-palmos-coff cross compiler, with the same flags as for the normal Linux gcc:

```
m68k-palmos-gcc -O2 -o hello hello.c
```

The result of this command is an m68k COFF object file named hello. The m68k COFF object file should be combined with the resources just created using the pilrc program. To complete this mission, we first use the **obj-res** utility that splits the COFF file into the code and data resources, and then use the build-prc utility to combine resources.

```
m68k-palmos-obj-res hello
```

This command will split the hello file into three resource files: code0000.hello.grc, code0001.hello.grc and data0000.hello.grc. Now we are ready for the final step of building the prc database.

```
build-prc hello.prc "hello" hwld *.grc *.bin
```

This command builds a prc application with a creatorID **hwld** and type **appl**.

Testing the Application

Testing the application on your Palm Pilot is not such a good idea. A bug might require us to reset the machine, and programs can't be debugged with the normal tools. The best way to test our just-created “Hello World” application is with a program that will emulate the Palm Pilot from within our desktop. POSE, available from http://www.palm.com/ with source code included, is a very good emulator. It can use a special debug ROM, also available from that site. With the debug ROM, we can get more debugging information on our application, which makes finding errors much easier, and we can hook gdb to it in order to debug with this well-known debugging tool. To install POSE, you will need the FLTK X toolkit. All instructions come with the POSE tar file.

There is one more emulator for X, called **xcopilot**. In order to use your own machine RAM or ROM, you can use the pilot-link package. The pilot-link package gives you the ability to communicate with the Palm Pilot from the serial port. There are some good utilities in this package, available from ftp://ryeham.ee.ryerson.ca/pub/PalmOS/. Using the pilot-link utilities, we can also transfer databases from our desktop to the Palm Pilot and from the Palm Pilot to the desktop.

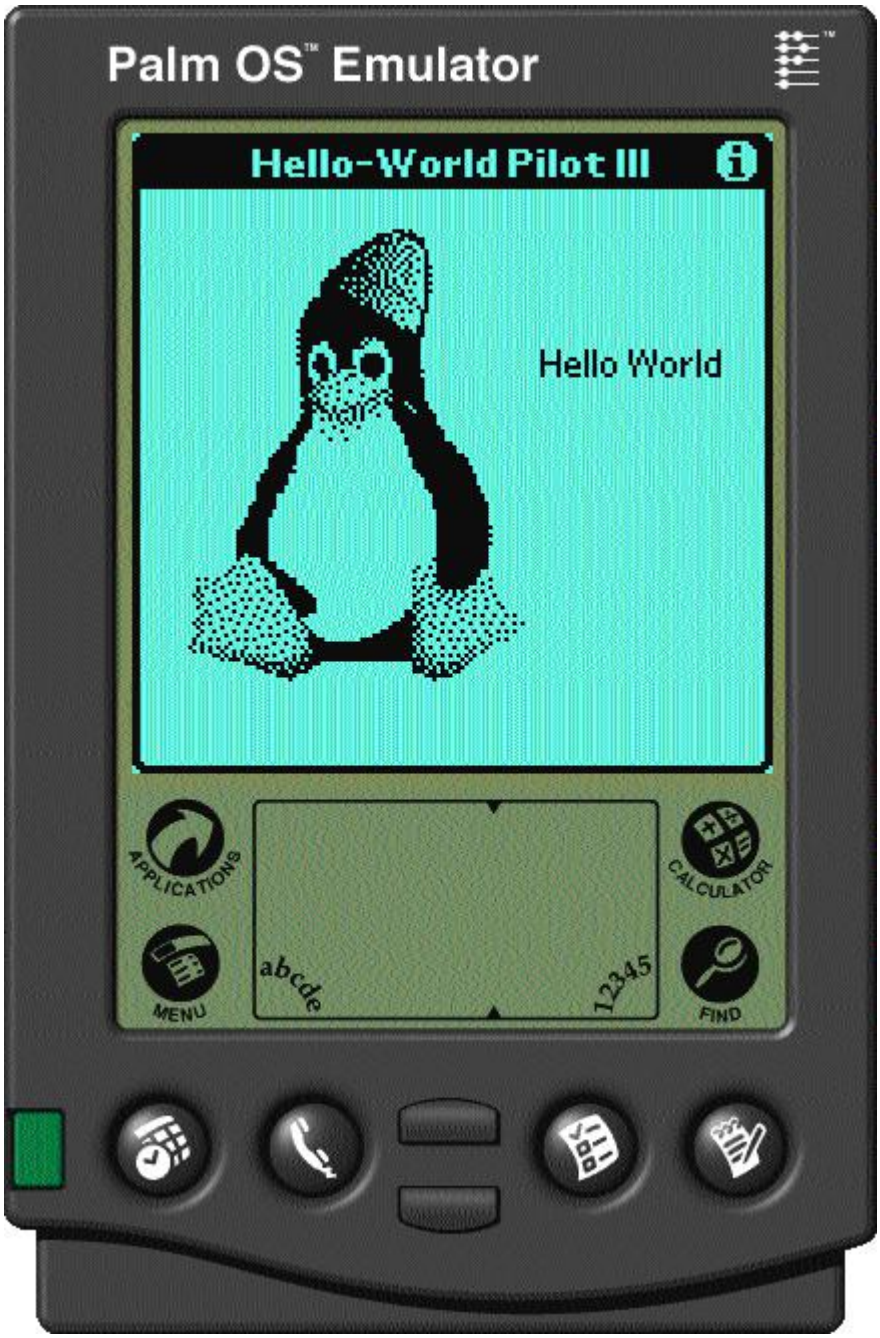


Figure 1

Figure 1 is how POSE looks with our **hello world** application for the Palm.

Resources

> Eddie Harari (eddie@sela.co.il) works for Sela Systems & Education in Israel as a senior manager and is involved in some security projects. He has been hacking computers for 13 years.

All listings referred to in this article are available by anonymous download in the file ftp.linuxjournal.com/pub/lj/listings/issue73/3782.tgz.

Archive Index Issue Table of Contents

Advanced search

Interview: Daryl Strauss, Precision Insight

Steven Pritchard

Issue #73, May 2000

Want to find out what's happening with 3dfx graphics hardware and the port to Glide? Read on.

Want to find out what's happening with 3dfx graphics hardware and the port to Glide? Read on.

The last time we talked to Daryl Strauss, he was working for Digital Domain, helping them generate graphics for the movie *Titanic*--that was February 1998. Daryl has since moved on to work for Precision Insight, where he has continued to work in the area of 3-D graphics. Steven Pritchard talked to Daryl about his current projects at the end of last year.

Steven: First of all, for those who don't know you, can you give a little bit of background on yourself? Specifically, how did you get involved in porting Glide to Linux, how long ago was that, and what exactly is your job description now?

Daryl: I started working on the 3dfx hardware just about three years ago. When the original Voodoo graphics came out, I looked at the specs and noticed that the performance numbers were very close to the original SGI reality engine. Since I knew the reality engine was a box that you could do real work on, I knew the Voodoo Graphics would be a very capable card. It literally changed the way the PC industry did 3-D graphics.

I ran out and bought an Orchid Righteous 3-D for $300 when they first became available. I wasn't disappointed. 3dfx has made the developer kit available to anyone from the very beginning. I downloaded it and noticed they had mentioned Linux in the documentation as being TBD. There were also a few **#ifdefs** in the headers. So, I knew someone there understood Linux. I went ahead and applied to their developer program to port the libraries to Linux.

It took about six months to work out all the details and about two weeks to do the first port. I think they were sort of surprised it happened so quickly, because there was another delay before the first version got posted on a web site.

Since then, I've continued to support the rest of their boards. This was always done as a volunteer effort in my spare time. Sometimes my real job got in the way. The work on the Voodoo Banshee took me a long time to get done, because it required writing an X driver (which I had never done before) and I was very busy creating the special effects for *Titanic* while that was happening.

Back in April, I realized that Linux was ready to have a large impact on the 3-D graphics industry. I left my job as software manager at Digital Domain and joined Precision Insight. My title there is Visual Mason and Evangelist. In June, Precision Insight signed a contract with 3dfx to develop a DRI implementation for their Voodoo Banshee and Voodoo 3, 4 and 5 hardware.

Steven: What exactly was just released today? What cards are supported? When do you anticipate a “production” release?

Daryl: This is a pre-release of an implementation of the OpenGL API for the Voodoo Banshee and Voodoo3 based on Precision Insight's Direct Rendering Infrastructure. That's quite a mouthful. What that means is users can run applications written for OpenGL in a window on a Linux system with a Voodoo Banshee or Voodoo3.

Another key feature of this release is the entire implementation is open source. Now that the basic capabilities are running, we're going to set it up as a public project so that anyone who wants may contribute.

It is still a pre-release, which means we know it still has bugs. We're getting it out there so that people can start trying to run their applications and see how they behave. We want to identify any issues. The final release will be in 2000.

Steven: Do you know if 3dfx takes Linux seriously, or if they just consider it a niche market?

Daryl: I can't speak for 3dfx. They were the first, and one of the few companies, to put money into a fully open-source solution for Linux. That's a big level of commitment. Some companies have released some code and some have released some specs, but few companies are making code and specs available and putting money into the development.

With that said, Linux is somewhat of a niche market today. Linux systems represent a very small number of sales. 3dfx realizes this is changing quickly. They want to get involved early so they can catch Linux as it grows.

Steven: How much do you know (or rather, how much can you tell me) about the new chip set and boards that 3dfx just announced? Are Linux drivers being developed? If so, will they be ready when the boards ship?

Daryl: 3dfx has published a lot of material on the new Voodoo4 and Voodoo5 chip sets. The key component is what they call the Voodoo Scalable Architecture. The first chip is the VSA-100. You can put between one and thirty-two of these chips together on a board. The fill rate on each chip (the number of pixels that can be drawn on the screen in a second) is a very impressive 333 megapixels. They've also improved the capabilities of the chip from that of the Voodoo3; it does 32bpp, stencils, 2048x2048 textures, texture compression, more color-blend modes, and a few powerful features called the T-buffer which allows full-scene hardware anti-aliasing. These boards will be very compatible with the OpenGL API.

Precision Insight will be working with 3dfx to support the Voodoo 4 and Voodoo5 boards. At this point, we haven't announced a release date for that software.

Steven: The specs on the Voodoo5 6000 look fairly impressive. Do you know if 3dfx is still just targeting gamers, or are they planning on targeting more “professional” 3-D users?

Daryl: The distinction between “gamers” and “professionals” is sort of artificial. What really matters is the sort of application being run. There are professionals who develop simulations (such as flight simulators or arcade systems) for whom these will be excellent boards. Quantum3D is a partner with 3dfx and will sell high-end configurations of the VSA-100 chip for this sort of user.

The real question is whether your application is fill-rate limited or triangle-rate limited. Digital content creation systems (CAD, 3-D modelers, etc.) are more likely to be triangle-rate limited. They have to draw many very small triangles. On these boards, all that work is done on the host CPU, and even with the very fast CPUs now available, it is likely that you couldn't push them through to the

graphics board fast enough to keep it busy. If your application draws bigger triangles or needs to do a lot of work with multiple-texture maps, it might be able to take advantage of the extra fill rate in these boards.

So, to answer your question, yes, I think these will be reasonable cards for any application, particularly with the new features they've added to the architecture. But, 3dfx is targeting the consumer, and that's where these boards will work the best.

Steven: What brought on the sudden decision to open the source to Glide?

Daryl: You'd have to ask them for a real answer. From my discussions, it is mostly that they want to have their hardware supported on as many platforms as possible, and open sourcing Glide seemed like the best answer.

Steven: What license is 3dfx using for the Glide source? Will it be possible to include it with the Mesa (or X) source?

Daryl: They made their own, but my reading was that it is basically LGPL. That means it can't go into X. It could go into Mesa, but I'm not sure we truly want that. It makes a reasonable stand-alone package.

A different, more ambitious project would be to remove the dependency on Glide from Mesa. That's a big project, but it might be interesting for people to pick up if they want to learn more about writing drivers.

Steven: Are drivers for any of the extra features, like the TV tuner on the V3 3500, being developed?

Daryl: Not by us. I believe they want to release the specs on them. The TV parts are a standard one for which you can get the data sheet. 3dfx will be releasing more specs fairly soon, so that may have the TV details. I'm not sure.

Steven: Was Precision Insight contracted to work on the Voodoo drivers on an ongoing basis or just for the initial implementation?

Daryl: We have a contract that includes work on the V4/V5 and to do some work on future boards. We may see more work after that, but this contract covers us for a reasonable period of time. We're not being contracted to do the end-user support. 3dfx will be doing that themselves.

Steven: What other companies have contracted Precision Insight to write DRI drivers? Are you working on drivers for any other cards?

Daryl: The only companies that have announced anything publicly are 3dfx, ATI and Intel.

Steven: Keep up the good work, and thanks for talking to us.

email: steven@silug.org

Steven Pritchard has been using Linux since 1993. He started both the Southern Illinois Linux Users Group (http://www.silug.org/) and the Linux Users of Central Illinois (http://www.luci.org/). He can be reached at steve@silug.org.

Archive Index Issue Table of Contents

Advanced search

Building a Wireless Network with Linux

Billy Ball

Issue #73, May 2000

Want your laptop and PC to talk to each other without having to deal with wires? Here's how.

Wireless networking has been around for a while, but until recently, it was out of reach for Linux users with modest means. Fortunately, a number of manufacturers are beginning to see new venues in marketing inexpensive wireless hardware. One manufacturer, Webgear, Inc., recently introduced a Linux-compatible wireless networking kit.

In this article, I will show how to create an inexpensive wireless network and a wireless networking bridge using Webgear's Aviator 2.4 wireless networking kit, IP masquerading, software routing, a surplus laptop and Linux.

Webgear's Aviator 2.4 wireless kit consists of two Type II PCMCIA cards and two ISA PCMCIA card slots. The kit uses 2.4GHz radio technology to offer 2MBps networking using IEEE 802.11 specifications (frequency-hopping, spread-spectrum networking). The cards appear as Ethernet devices, such as eth0 or eth1, when installed and configured. The original intent of the kit is to offer the ability to create a high-speed wireless link between two desktop PCs, two laptops, or a laptop and desktop PC.

Although the kit is advertised as supporting Linux, it comes with software drivers for only Microsoft operating systems. You'll have to download the Linux device driver, a loadable kernel module named ray_cs.o, from the Web. Even though you can get a stable version through Webgear's support link at www.webgear.com/support/software_top.html, the best place to download the latest version is from the author's web site (see Resources).

Thanks to the generosity of the author, Corey Thomas, you can use the Aviator 2.4 (or more expensive Aviator Pro or Raytheon Raylink series) wireless system

with Linux. I downloaded the latest driver, **ray_cs** version 1.68, onto one of my laptops, then followed his directions on building and installing the driver:

```
cp ray_cs-1.68.tgz /usr/src/linux/pcmcia-cs-3.1.5
tar xvzf ray_cs-1.68.tgz
make config
make all
make install
```

These steps should then be repeated on another computer. I initially used two laptops to test the connectivity of the cards. This is essential to ensure the cards and software are working correctly. I had great success with a later version of David Hinds' Card Services, pcmcia-cs 3.1.5, along with the next version, pcmcia-cs 3.1.6.

Peer-to-Peer Wireless Networking

As sold, the kit allows for connection between two wireless points: point A**-->**point B.

Before enabling the cards, you must first edit /etc/pcmcia/config.opts and insert the line

```
source ./ray_cs.opts
```

Upon starting, Card Services will then read in the ray_cs.opts configuration file for the wireless cards (ray_cs.opts is copied into the /etc/pcmcia directory as part of **make install**). This file contains several important settings, in the form of setup strings, used to configure the Aviator 2.4 card after insertion. The settings line (there are several, as the driver supports at least three different wireless cards) I used was:

```
module "ray_cs" opts "pc_debug=2 essid=LINUX\
 hop_dwell=128 beacon_period=256 translate=1"
```

Make sure to use the same settings for each end of the wireless connection. The **pc_debug=2** option is a handy way to get more information from /var/log/messages upon insertion or removal of the card. The **essid=LINUX** option designates a network name of LINUX for the wireless cards (according to Webgear, 61 cards may be on the same network). After installing and configuring the software, I used Red Hat's **netcfg** tool to create an eth0 interface on each laptop. You can also create your own by editing the /etc/sysconfig/network-scripts/ifcfg-eth0 file:

```
DEVICE=eth0
IPADDR=192.168.2.37
NETMASK=255.255.255.0
NETWORK=192.168.2.0
BROADCAST=192.168.2.255
ONBOOT=yes
BOOTPROTO=none
USERCTL=no
```

The cards must be started sequentially. I went to the first laptop, ensured Card Services was running (you can do an /etc/rc.d/init.d/pcmcia start), then inserted one of the Aviator cards. Upon hearing two beeps, I looked at the output of /var/log/messages (not all the output is shown here):

```
$Id: ray_cs.c,v 1.68 1999/11/21 10:43:35 corey Exp
$-Corey Thomas
corey@world.std.com
ray_cs Detected:
WebGear PC Card WLAN Adapter Version 4.88 Jan 1999
eth0: RayLink, irq 4, hw_addr 00:00:8F:48:E8:DB
ray_cs interrupt network "LINUX" started
```

The important thing to look for is the “started” string. I then enabled the interface using netcfg, as according to Thomas, you cannot use the **ifconfig** command to set the IP address using the ray_cs driver. I then went to the other laptop, inserted the other Aviator card, listened for the two beeps, and looked at /var/log/messages:

```
$Id: ray_cs.c,v 1.68 1999/11/21 10:43:35 corey Exp $-Corey Thomas
corey@world.std.com
ray_cs Detected:
WebGear PC Card WLAN Adapter Version 4.88 Jan 1999
eth0: RayLink, irq 10, hw_addr 00:00:8F:48:E8:45
ray_cs interrupt network "LINUX" joined
```

Again, the important string to look for is “joined”. This means the cards are communicating. I then enabled the eth0 interface, and voil<\#224>--I could use **ping**, **telnet**, **ftp** and **talk** between the two laptops wirelessly. FTP file transfers were about 180K per second, and that speed is reasonable enough for remote X11 client launching. This shows that Linux can be used to support wireless networking in a variety of situations where cabled installations are impractical or unnecessary.

Wireless Masquerading from a Server

While being able to network two laptops wirelessly is a lot of fun and can be quite handy, I was still not satisfied. I have a local area network (LAN) in the house, spanning several computers hubbed together in the basement and up two levels to an upstairs office with another hub of three computers. Being a Linux user, I want more from my hardware and wanted to get much more functionality out of my wireless network. I wanted to be able to roam the house or sit by the pool with a wireless laptop, and also network with all computers on the LAN while browsing the Web, doing e-mail and so on. The cards are advertised as being able to communicate at least 500 feet through walls and 1000 feet within “line of sight”.

Unfortunately, according to Webgear, the Aviator 2.4 cards are meant to be used only for peer-to-peer networking. If you purchase the Aviator Pro line of cards, you can then buy a piece of hardware called a wireless “access point” that hooks into your LAN via RJ45 to provide a bridge into your network or from another network. The access point costs several hundred dollars, even with

diligent Internet shopping. A wireless access point is not offered for the Aviator 2.4 series.

This is where being a member of a Linux users group comes in handy. Thanks to friends in the Northern Virginia Linux Users Group, NOVALUG, I was able to create my wireless network. Following tips from messages on NOVALUG's mailing list, I installed one of the Aviator ISA PCMCIA slot adapters in my server in the basement. The server's first interface, eth0, has an assigned IP address of 192.168.2.XX. I installed the Aviator card as eth1 with an IP address of 192.168.1.1. After the wireless card was “started”, I then used the **route** command to route wireless (eth1) traffic through eth0:

```
/sbin/route add 192.168.1.1 gw 191.168.2.
```

To complete the services, I then crafted a simple IP masquerading script (based on information courtesy of Greg Pryzby, NOVALUG's founder):

```
#!/bin/sh
case "$1" in start)
      /sbin/modprobe ip_masq_ftp
/sbin/ipchains -A forward -s 192.168.2.0/24 -j MASQ
/sbin/ipchains -A forward -s 192.168.1.0/24 -j MASQ
      echo "NAT Started" ;; stop)
/sbin/ipchains -F
      echo "NAT Stopped" ;; *)
      echo "Usage: ipmasq {start|stop}" ;;
esac
```

After enabling my PPP connection, wireless card, masquerading and routing on the server, then connecting to the rest of the LAN and using the Internet from my wireless laptop (with an assigned IP address of 192.168.1.2) was quite simple. Again, using the route command, but this time on the wireless laptop:

```
/sbin/route add default gw 192.168.1.1
```

This configuration allowed me to communicate with all the other computers on the LAN (such as LaptopA and ComputerB), along with access to the Internet.

Figure 1. Network Configuration

Wireless Masquerading from a Laptop

Then I ran into trouble. When I was down in the basement near the server, connection tests ran okay with the wireless laptop. However, when I ranged too far from the server, for some reason the cards would lose communication, or I could not “join” the “started” server connection. If you read more about wireless networking's coverage and reliability, you'll learn that the quality of a connection can be influenced by many factors. I later found out the equipment in the basement was located too close to an earthen wall, and for some reason,

the signal was being blocked. I had to move the connection, but I still wanted full access to the LAN and the Internet.

I then removed the Aviator card from the server and reinstalled the card on an ancient spare laptop (LaptopA) upstairs with two Type II PCMCIA slots. Like the server, one laptop slot provided an eth0 interface to the LAN (with an IP of 192.168.2.32), while the other was configured as eth1 with the wireless card (and an IP of 192.168.1.1). IP masquerading on the server was forwarding packets from the PPP connection to the laptop, but in order to provide service to the wireless laptop, I had to use IP masquerading on LaptopA again:

```
/sbin/ipchains -A forward -s \
192.168.1.0/24 -j MASQ
```

Now everything worked fine! I could communicate with the rest of the LAN (such as from the Wireless to Computer B or the Server).

A simple route command on the server, using the wireless laptop's IP address along with the "hardwired" eth0's IP address (from LaptopA) also allowed the rest of the LAN to ping the wireless computer:

```
/sbin/route add 192.168.1.2 \
gw 192.168.2.XX
```

Believe me when I say it was somewhat of a learning experience—I never envisioned having to forward information from two computers.

Bridging: An Alternative

I found out you also can easily build a wireless bridge, and thus not have to use routing to allow a wireless connection to connect to the rest of a LAN. Using a spare laptop, I first recompiled Linux to enable bridging. I then installed an eth0 interface with an assigned IP address to connect to the LAN. Next, I installed the Aviator card as eth1 *without* an assigned interface, then brought up both interfaces in promiscuous mode with

```
/sbin/ifconfig eth0 promisc up
/sbin/ifconfig eth1 promisc up
```

Next, I downloaded Alan Cox's **brcfg** utility, and enabled bridging with

```
./brcfg -ena
```

After starting a wireless connection, I could then access any computer on the LAN from the wireless laptop.

Conclusion

Wireless networking may not be the best solution if you need high-speed communication on or between your LANs, but the combination of Linux and a legacy laptop shows that you can build a useful and flexible wireless network at low cost. This is just one of the reasons I use Linux (besides being able to surf the Web while drinking a pool-side Margarita—with salt, on the rocks, thank you).

Resources

Acknowledgements

Bill Ball is a member of the Northern Virginia Linux Users Group (NOVALUG), and the author of nearly a dozen books about Linux. He may be contacted through http://www.tux.org/~bball/.

Archive Index Issue Table of Contents

Advanced search

Linux Webpads Give PC Competition

Linley Gwennap

Issue #73, May 2000

New hardware for a new generation.

The unveiling of Transmeta's Crusoe processor (see last month's article “Transmeta Writes the News”) highlights an emerging class of computing devices, often called webpads. These devices provide high-quality web access in a conveniently portable design. Resembling the display portion of a mobile PC (see Figure 1), webpads have no bulky keyboard or heavy disk drives. Better yet, they don't need Microsoft Windows—Linux or other low-cost operating systems are preferred.

Webpad Prototype

These devices are only beginning to reach the market, and with prices as high as $999, they will appeal mainly to front-line soldiers in the gadget wars. But prices will inevitably fall, and at under $500, webpads will offer a great alternative to those who want the Web without the hassle of a Windows PC. Although webpads don't have to use Linux, its reliability, modest memory footprint and especially its open source code make it an excellent choice.

Why Webpads?

Webpads offer several advantages over a PC. The first is ease of use. A webpad does one thing: browse the Web. With a dedicated browser on top of a stripped-down version of Linux, the system provides a simple, reliable interface with minimal boot time. Microsoft offers Windows CE for such “information appliances”, but that OS still bears the scars of its Windows heritage.

Webpads offer mobility that can't be matched by a PC. All sub-$1,000 PCs today are desk-bound. Even if you pay more to get a notebook PC, you get a device weighing at least three pounds, and more typically, five to seven pounds. This weight is mostly due to hard drives, CD drives, large batteries and cooling mechanisms (fans and heat sinks) for the Intel-compatible CPU.

A webpad weighs only about one pound and has a longer battery life, despite using a smaller, lighter battery. Like a cordless telephone, it links to a base station using radio frequencies. Thus, a webpad will operate anywhere within a typical house or small office. The wireless link transfers data at 2 to 11MBps—enough to keep up with even a DSL Internet connection. Eliminating the drives gives webpads a sizable cost advantage, at least when compared to mobile PCs. In contrast to the cheapest desktop PCs, webpads will cost more, because their flat-screen displays are more expensive than CRTs and also due to the cost of the wireless link.

Today's PCs also benefit from economies of scale. Within a few years, increasing volumes and lower-cost screens should bring a webpad under $500, the price of the least expensive PC with monitor. At this price, webpads should greatly increase in popularity.

Other non-PC devices already provide web access, including WebTV, Sega's Dreamcast and the Palm VII. WebTV and similar devices use a standard low-resolution TV as the display, so they can't display a full web page from most sites without scrolling. And, of course, they aren't mobile. The Palm VII uses cell-phone technology to provide unmatched mobility, but its display has even less resolution than a TV, limiting it to web sites designed specifically for the hand-held device. A webpad, in contrast, will typically have a VGA or better color LCD display comparable to a 14-inch monitor, providing full compatibility with any web site. The Palm VII also requires a monthly cellular subscription fee; the webpad does not.

Web Model Threatens PCs

Of course, webpads are not PCs and aren't compatible with the vast range of PC software. But they are good for accessing a variety of services over the Web, including news, shopping, e-mail, chat, banking, voice mail and whatever the

dot-coms think of next. In fact, the plethora of new web services is undermining the value of the PC's flexibility. You used to need a PC if you wanted to log your appointments, maintain an address book, track your stock portfolio and calculate your income taxes. Now all these services are available on any platform with a web browser. Still, webpads won't replace most PCs in the foreseeable future.

Using an on-screen keyboard or handwriting recognition, webpad users can enter URLs and short e-mail messages, but these interfaces are not ideal for creating memos, papers, spreadsheets, drawings and other documents needed for school, work and even some personal uses. High-quality voice recognition could begin to close the gap, but that is still years away. In short, webpads are ideal for viewing and interacting with information, while PCs remain an efficient platform for creating content. PCs are likely to stay on most business desktops and in many homes for this reason. However, high-income households may someday contain a single PC and one or more webpads instead of multiple PCs.

The real opportunity lies in penetrating the households that don't have a PC today: 50% of the U.S. market and far more in other regions. Many of these consumers can afford a PC, but don't want to deal with the complexity of that platform. By eliminating Windows, a Linux-based webpad solves this problem. Even if they don't displace PCs, webpads could one day outsell them.

email: linley@linleygroup.com

Linley Gwennap (linley@linleygroup.com) is the founder and principal analyst of The Linley Group (www.linleygroup.com), a technology analysis firm in Mt. View, California. He is a former editor in chief of Microprocessor Report.

Archive Index Issue Table of Contents

Advanced search

Igel Etherminal J and Instant TC

David Weis

Issue #73, May 2000

One of the many uses of Linux is creating thin clients to save time and money.

- Manufacturer: IGEL LLC
- E-Mail: sales@igelusa.com
- URL: http://www.igelusa.com/
- Price: $480-$699 US
- Reviewer: David Weis

One of the many uses of Linux is creating thin clients to save time and money. Products in the Etherminal line of thin clients are built with that in mind. They are available in a variety of configurations, including Java support, X support, text-based terminal emulation and Microsoft Windows terminal support.

An innovative feature of these products is the fact that the entire file system is stored in flash memory. This will eliminate problems with improper shutdowns and the short lifespan of spinning magnetic media in harsh environments.

Features

I tested the Etherminal J and the Instant TC (thin client) products. The Etherminal J includes Netscape Communicator with Java support, the Citrix ICA client, an RDP client, an X server, terminal support and printer support. The Instant TC is a kit that includes these features, but comes on an 8-bit ISA card with an included S3 video adapter.

The Etherminal J is a turn-key unit that is 11-1/2 inches wide by 8-3/4 inches deep by 2 inches tall. It includes mounting feet to orient the unit vertically to save desk space. It's powered by a 180MHz Cyrix processor with 32MB of RAM, on-board 10/100 Ethernet, 16-bit sound, S3 Video, two serial ports, one parallel port and a PS/2 keyboard and mouse port. You need to supply the monitor, keyboard and mouse.

Installing the Instant TC into a machine is quite easy, consisting of installing the two cards and a supported network card: only ISA and PCI NE2000 cards and the Intel EtherExpress Pro.

Good Points

Both products performed well in daily use. I used the Etherminal J as my desktop machine for two days in order to get a good feel for the performance.

A full version of Netscape was included with both products, and it was more stable than the same version on my normal machine. Once Netscape had loaded, it was very snappy and performed well.

The Etherminal included terminal emulation support for SCO-ANSI, Linux-ANSI, VT320, Wyse60 and SNI97801. The function keys worked correctly when connecting to Linux servers, in addition to an RS/6000 and an AS/400 to which I have access.

One use for the Etherminal is as an X terminal. You can easily set up XDMCP to connect as a client to a larger machine or run X applications remotely and have them appear on your local display. With a 100Mbit network, the response was as good as running them locally.

If you need access to Microsoft products, the Etherminal has the capability to connect to the Windows Terminal Server and the Citrix Metaframe Server. I don't have access to either of these, but the clients are the same ones available from Citrix, so I have no doubts as to their functionality.

The Good/The Bad

One pleasant surprise was how nice the fonts looked on the Etherminal. They were nice, sharp 100dpi (dots per inch) fonts that were very easy on the eyes. Selecting fonts for your applications is made easy by the Igel Setup Program.

After starting either of the devices, you will arrive at an X desktop. From the menu on the screen, you can run the Setup Program which allows you to configure all aspects of the client software including the keyboard, language, network, serial ports, applications, time and date. Updating the firmware is also done using the Setup Program.

The Setup Program is one place where the amount of work put into this product is evident. Changing the setup is quite easy and intuitive. Selecting a screen resolution takes only two clicks. Adding remote applications to the menu is also easy. You need to choose whether the remote host will be accessed using a text emulation, a Windows Terminal emulation or via X. For text emulation, there are selections to specify window background and foreground colors, font size, cursor size and the size of the window in rows and columns.

Documentation included with the Etherminal consisted of a 92-page manual covering the Etherminal C, W and J in English and German and a 42-page manual for the actual hardware. The Etherminal manual contains detailed information on how to configure your Etherminal. The hardware reference manual covers BIOS setup. However, you shouldn't need to modify any of the BIOS parameters from their defaults.

In the course of my evaluation, I did find a few problems with the Etherminal. Flash memory is much slower than most hard drives. Netscape Communicator took over 60 seconds to load. A dialog box did appear after launching Netscape, to reassure you it was starting up. Closing and reopening Netscape got that time down to under five seconds, showing that it had been cached in memory.

Another point that isn't as obvious is that the flash memory driver used for the Linux kernel on the Etherminal isn't released under the GPL and is statically linked with the kernel. However, Igel was the original author of the wrapper to link the M-Systems driver with the kernel, and Igel has released the portion they wrote.

Conclusion

The Etherminal and Instant TC are both well-built and well-designed products. They are excellent replacements for dumb serial terminals, X terminals and older PCs. I have no problem recommending them.

Update

David Weis (djweis@sjdjweis.com) is a programmer at Perfection Learning Corporation in Des Moines, Iowa. He enjoys spending time with his wife and friends and writing free software.

Archive Index Issue Table of Contents

Advanced search

NetTEL 2520/2500, PoPToP

Jon Valesh

Issue #73, May 2000

A NetTEL does nothing a network whiz couldn't do with their favorite Linux distribution, an old PC and a network card or two, but with a NetTEL, you don't need to be a network whiz.

- Manufacturer: Moreton Bay
- E-Mail: sales@moretonbay.com
- URL: http://www.moretonbay.com/
- Price: under $400 US
- Reviewer: Jon Valesh

There are three words that go to the heart of Linux's appeal for many of us, words that define the spirit of Linux developers and users alike: “Do It Yourself”. It could be the Linux motto, but even with that great do-it-yourself spirit, there are times when having someone do it for you sure would be nice.

The new NetTEL VPN routers from Moreton Bay are great examples of how nice it can be. A NetTEL does nothing a network whiz couldn't do with their favorite Linux distribution, an old PC and a network card or two, but with a NetTEL, you don't need to be a network whiz. You can forget about understanding configuration files, and just tell the NetTEL what you need it to do. If you are feeling like a whiz, you can still learn those configuration files and do it yourself if the need or fancy takes you, because the NetTEL runs Linux and uses the same GNU tools you would have used to do the job.

Let's forget that attractive fact for a moment and concentrate on what the NetTEL does. The NetTEL is first and foremost a Virtual Private Network (VPN)/ Network Address Translation (NAT) router. The 2500 model includes one Ethernet interface and two serial ports for modem or ISDN Internet access or incoming PPP connections to your LAN. The 2520 model adds a second Ethernet interface for ADSL or cable Internet services.

For its VPN role, the NetTEL uses the widely available point-to-point tunneling protocol developed by Microsoft to allow secure access to your local area network (LAN) by computers anywhere on the Internet.

As a NAT firewall router, the NetTEL allows every computer on your LAN to share a single Internet IP address, protecting your computers from Internet attacks, saving you money and allowing you to share resources more easily.

The Good, The Bad

The NetTEL doesn't stop there, of course. Moreton Bay obviously made a real effort to cram as much network functionality into the NetTEL as they could. A DHCP server, dial-in support, transparent IP tunneling, configurable packet filtering, a good suite of diagnostic tools and more merge to form a very useful network tool for small to medium-sized businesses as well as home users who want the latest gadgetry.

Using a NetTEL, you can access the Internet from all computers on your LAN without paying for extra IP addresses or using expensive dedicated Internet services. That's because the NetTEL supports NAT, a feature derived from the Linux kernel networking support. NAT has one unchangeable rule: you can reach the outside world, but the outside world can't reach you. Great for security, but lousy if you want to transfer a file from your office computer to one at home. Moreton Bay has addressed this by implementing VPN, which allows you to create network links between any computer running the proper software and your LAN. Thus, you can access computers sequestered by the NAT firewall without exposing them to the hazards of being on a public network.

If you don't always have Internet access or you want to link remote offices without sending your data over the Net, the NetTEL's dial-in PPP feature allows modem-based wide area networking between NetTEL's or any PC or router supporting dial-out PPP. Incoming PPP and VPN connections are user ID- and password-authenticated, using information stored in the NetTEL. All that can be configured by anyone, even if they have never set up a network before and don't know VPN from PNG—that's the promise, anyway.

As with everything, the ultimate test is getting everything running the first time. A smooth start is doubly critical for a product like the NetTEL. After all, if it isn't easier to install than building your own, what's the point?

The NetTEL I received was a pre-release version shipped before the final packaging was ready, so a few details like a printed quick-start guide were omitted. The retail package should be a little easier to get working.

Documentation

If you have some real work to do and no time to learn a new trade just to set up a network, the NetTEL's manual will come in very handy. You can download it in PDF format from the Moreton Bay web site to check out the installation and configuration process before spending any money. The manual doesn't limit itself to details on setting up the NetTEL, either, so if you are new to LANs and especially VPNs, you'll get a good introductory tutorial on the technologies and configuration of both to get you started. Die-hard Linux nuts will find the manual a bit too Windows-oriented. Fortunately, the instructions assume you will be using the web interface for configuring all the major features, and that keeps it platform-independent.

Installation and Configuration

You configure and maintain the NetTEL through a slick web-based user interface, although the router ships without an assigned IP address which makes accessing those pages somewhat difficult to start. Before you can configure anything, you must assign the NetTEL an IP address on your network. Since you cannot connect a keyboard, monitor or serial console to the NetTEL, you need a working LAN with at least one computer on it to convince the NetTEL to do more than flash its lights dolefully.

The good news is you don't need to find a Windows machine or run any NetTEL-specific software to assign the NetTEL its address. You can use a standard DHCP server on Linux or Windows. Needless to say, this is going to be a problem for some users. If you already have enough of a network to be running a DHCP server, your network is probably enough that you don't need the NetTEL. On the bright side, setting up a DHCP server is less work than duplicating the features of the NetTEL, so it is still a fair trade. If you have a Windows machine around, an install wizard will transfer the spark of life to the NetTEL in minutes.

Once an IP address has been assigned, configuring the NetTEL is easy. Bring up your web browser, enter the IP address, and a nicely designed web page will guide you through setting up the Internet connection type, creating dial-in and VPN connections, dial-out Internet connections, configuring the internal DHCP server, security filtering, etc. If you are using the NetTEL for dial-up Internet access, you can configure it to remain connected all the time or automatically establish a dial-up connection whenever you go to access the Net. The configuration options are explained well on the web page, and the manual includes item-by-item instructions for each configuration page.

If you already have a DHCP server or use the Windows software, the complete installation and configuration can take less than 20 minutes.

Operation

In operation, the NetTEL usually does exactly what a router should—disappears into the background. It does its job, and you don't have to worry about it. If you do run into trouble, there are diagnostic tools available on the NetTEL's built-in web page. You can even set up the NetTEL to log error and status messages to your Linux machine using **syslog**, eliminating the need to check the web page for error messages.

When talking about network equipment, reliability is king. One of the strongest advantages of the NetTEL is the inherent reliability of its compact, solid-state design. No hard drive to trash, no fan to stop spinning, no keyboard to spill Coke on and no monitor to drop on your foot. It just silently does its job, and that's something very hard for a PC-based solution to compete with. I had no problems with the NetTEL hardware.

Moreton Bay put some thought into field-upgrading the NetTEL, too. The firmware can be upgraded using either **tftp** to a Linux machine or by using a Windows utility that is probably just a smart tftp server. Upgrading the firmware from a Linux box can be as simple as clicking a button on the web page. After an upgrade, the NetTEL will remember your old configuration options if possible.

Performance

It is all very fine to talk about sharing a single Internet connection, but sharing means slow, right? Not always, as I found out. I was amazed to see about a 20% throughput improvement with the NetTEL over dialing directly from my Linux desktop computer to access the Web. Fast serial ports and a dedicated CPU can truly help network performance. I didn't notice any improvement when using the 2520's second Ethernet connection to access the Internet like an ADSL or cable connection, but I didn't see a noticeable slowdown, either.

Downside

Not everything is rosy. The NetTEL does its job very nicely, but the job it does comes with some limitations. When using the NAT feature to share a single IP address, the NetTEL acts as a rigid firewall, blocking all incoming connections to your computers. This is normally a good thing, but not all network applications will work when you cannot make an incoming connection to the client computer. Unfortunately, there is no guaranteed workaround for those programs, and there is no way around the NAT feature when using the NetTEL to connect to the Internet.

The VPN feature provides the workaround for NAT's obstinate blocking of incoming packets, in some cases. If you know you will need to interact with a particular machine outside your physical network, you can set up a VPN link to bring that machine into your virtual network. Unfortunately, that works only for people you know and machines you or they control.

If you use the dial-on-demand feature, the NetTEL automatically connects to your ISP whenever you go to bring up a web page or run an Internet application like TELNET, but it isn't very selective about what programs cause it to dial. If you have software that periodically tries to make a connection to a remote host, such as a network time server, your dial-up connection will either stay up 24 hours a day or connect at seemingly random times. I had been testing the NetTEL for about three weeks when I got an e-mail from my ISP saying that perhaps I should lay off the caffeine because I was constantly dialing, staying on-line for fifteen minutes, and disconnecting for another five, only to dial back up again. I traced that back to **xntpd** running on one of my Linux boxes, but I've seen the same interaction from several other programs, most of them Windows applications. If you are unfamiliar with **tcpdump** or some other network monitoring tool, tracking down unsolicited dialing can be a bit of a trick.

I also had a minor problem with the modem I used to test the NetTEL's dial-up networking support. The NetTEL's modem initialization script is very generic, but wasn't able to reset the modem and place a call. That's where the NetTEL's Linux core appeals to anyone familiar with Linux networking. A minor change to the same chat script you'd find on a full Linux PC, and everything was working great.

In the end, these problems are just proof that there are no magic bullets—they are almost all unavoidable side effects of the good features of the NetTEL.

Upside

As VPN/NAT routers, the NetTEL 2500 line is excellent. At less than $400, it makes a lot of sense for any but the most die-hard do-it-yourselfers in the market for a NAT router or VPN solution.

But there's more. The NetTEL isn't a total loss for the do-it-yourselfer. In fact, the NetTEL (actually, the circuit board which forms the heart of the it) is one of the neatest toys I can imagine a hard-core do-it-yourselfer getting their hands on. Under everything, the NetTEL runs Linux (uClinux, specifically) and that means source code. The uClinux source is fairly monolithic, with code for the kernel and every application in one tree, sharing a common Makefile. The Makefile builds a compressed ROM image, including file systems, kernel and your application software, ready to load. The standard source tree gives you

the network-heavy feature set of the NetTEL itself: **pppd/diald**, Ethernet support, the Boa web server, etc., missing only the NetTEL-specific web interface. From there, parts can easily be added or taken away. Don't expect the latest kernel features, though, as uClinux is based on the version 2.0 kernel, and the libraries have been trimmed with an eye to saving space over preserving functionality.

I must admit I looked over only the development tools. I didn't roll up my sleeves and start coding, but I did enough to learn that if you decide to use a NetTEL to build your own embedded Linux system, you will end up investing in some hardware debugging tools specific to the ColdFire CPU. The NetTEL hardware has a single FLASH ROM containing both the user-loaded code and the boot loader, so if you load a bad system image or in some way manage to disable the networking support, you will end up with a very attractive paperweight until you get those tools.

The Motorola ColdFire CPU is fast, and for those who decide to step below C for added performance, the commands and philosophy should be familiar to anyone who has worked with the Motorola 68K family. Some neat additions and odd changes are tossed in to keep it interesting. The serial ports support data rates of up to 230Kbps, and networking is harder to turn off than to leave on.

The memory and storage will seem vast to anyone used to microcontroller projects, but don't expect Emacs to fit. You've got 1MB of FLASH and 2MB of RAM, enough to do serious work. If you need more, upgraded versions are available from Moreton Bay. Size-induced limitations in the library may cause some ported software not to work without major changes, or to behave in strange and less than wonderful ways. For the most part, standard applications will port nicely to the uClinux platform.

Physically, the circuit board is a little larger than it needs to be because Moreton Bay left room for a PCI slot and support chips, yet the board is still smaller than almost any PC-based Linux solution. You cannot fault anyone for having too much expandability, and the ability to add a PCI card could actually change the development curve for some projects. The board has a very flexible power supply, able to accept AC or DC ranging from 6 to 12 volts, which should be perfect for running from a car battery or other alternative power source. The entire board draws less than five watts while running.

In talking with the folks at Moreton Bay, they are just as excited about the embedded applications for their design and uClinux as they are about the networking applications for the NetTEL. They have a lot of plans to add features specifically for embedded developers, and some special hardware packages are

available too, so it is definitely worth thinking about if you are starting to develop a Linux-based embedded system.

PoPToP

Just to prove once and for all that the folks at Moreton Bay really do understand the spirit of Linux and open software, there is one other tool for the do-it-yourselfer. PoPToP, the Linux PPTP server, was originally developed by Matthew Ramsey at Moreton Bay but released under the GNU GPL. It is currently supported by a team of people around the world and has been ported to several platforms besides Linux. With PoPToP, you can implement your own VPN server, so if you really like the idea of a VPN router but finances or philosophy won't let you buy a NetTEL, you can just download the PoPToP software and install it on your own Linux machine for free. You will find a link to the PoPToP web page on Moreton Bay's web site.

I installed the PoPToP software on an Intel Pentium II machine running a 2.2.*x* kernel, and while installation was fairly painless, it was a compiler job, so don't be surprised if you run into problems getting the software to work on your computer. You will end up with the source for pppd and PoPToP, and you may need to recompile your kernel to get the software to work.

Once configured, the PoPToP software allows you to connect remote computers to your network through tunneled PPP links. PoPToP's configuration files are almost identical to those for pppd, so if you have ever configured your computer to accept incoming PPP connections, you'll have no trouble with PoPToP. If you haven't, the PoPToP FAQ provides sample configuration files that will work with a minimum of change, and the PoPToP web page is full of great links to VPN resources. Once everything is working, connections behave exactly like any serial PPP link. You can route subnets or individual IP addresses through the tunnels just as you can with standard PPP connections. It does what you always hoped networking tools would do—it disappears, never to need fiddling with again.

Born at the beginning of the Microcomputer age, **Jon Valesh** (jon@valesh.com) has pushed and been pushed by computers his entire life. Having run the gamut from game programmer to ISP system/network administrator; he now occupies himself by providing technical assistance to ISPs and small businesses whenever his day job doesn't get in the way.

Archive Index Issue Table of Contents

Advanced search

Conectiva Linux 4.0

Jason Kroll

Issue #73, May 2000

Over 500 million people speak Spanish or Portuguese as a first language. Conectiva brings Linux to them.

- Manufacturer: Conectiva Linux
- E-Mail: info@conectiva.com
- URL: http://www.conectiva.com/
- Price: $79 US
- Reviewer: Jason Kroll

Over 500 million people speak Spanish or Portuguese as a first language. Conectiva brings Linux to them.

At the last COMDEX, I had the pleasure of chatting with the Conectiva folks who flew all the way up to Las Vegas from Brazil for the Linux Business Expo. *LJ* was very enthusiastic when we heard some time ago that Conectiva was translating every man page, system message and other documentation into Spanish and Portuguese, thus making Linux much more accessible to an enormous user base. It has been wonderful to see how far Conectiva has come in its endeavor to bring Linux to Latin America. Already there are several software packages, as well as an exciting new Brazilian magazine *Revista do Linux*.

Circumstances of continental drift have brought North, Central and South America fairly close together, and we have a lot of shared history, although not all of it is particularly positive. Nevertheless, one must put a regional perspective on the whole affair to appreciate the enthusiasm many of us have had for this project. Indeed, as Europeans often learn English in school, English-speaking Americans are often taught Spanish. Public telephones, ATMs, advertisements and more than a few signs are often bilingual (although this might be largely a West Coast and Southwest thing), so one cannot help but feel a kind of victory to see Linux become translated into not just Spanish but also Portuguese (which has typically been overshadowed by its more widely spoken relatives). In fact, Conectiva is actually Brazilian, so the Spanish translation was a secondary but much appreciated endeavor.

Latin America is a bit behind the U.S. in terms of keeping up with the latest in hardware, so on average, there are slower processors, less RAM and smaller hard drives, making it difficult and painful (if not impossible) to run the latest bloated proprietary wares. Additionally, most people do not have the money to waste on proprietary software, let alone expensive development kits and the requisite fancy computers. When you look at the situation in businesses which require several or even dozens of computers, and especially the typically less-than-well-financed universities, the price of licensing software is too high. Out of necessity, this has led to the choice between having enough computers and paying licensing fees for the software. When you would be paying licensing fees to multi-billion-dollar corporations in some foreign nation, the choice is fairly obvious. Even without these considerations, the choice is obvious—you don't pay the fees.

Unfortunately, every time money is involved with something, people start pointing guns at each other. The Conectiva folks told me stories of corporations being raided and executives leaving in handcuffs, as well as universities being harassed and intimidated. Apparently, it's a common occurrence (and we thought the “hang your boss” campaign was going way too far). Alas, the zealous persecution of software users in Brazil is driving businesses and universities away from proprietary software and toward Linux.

GNU/Linux is the ideal operating system for Latin America because it provides everything universities, home users and businesses need, all free of charge; it runs extremely well on minimal hardware; and, of course, all the source code is available and often well-annotated. A free C compiler alone is worth a fortune. One need merely peruse the directories at Metalab to see the wealth of free software (remember your first visit to sunsite or the GNU archives?). Higher quality, completely for free, and most relieving of all, no more living in fear of the police. You can buy one copy of Linux and install it on every computer in your lab. And you don't need anything more than a 486/33 (though the new

release will be Pentium optimized), so you can dig up any old PC you've got lying around and have one more for your lab.

Networking is a breeze; that's probably what Linux does best, and the free support from the community is priceless. Linux users have typically been most fond not as much of the free price but of the free source. Sometimes we forget that Linux also wins economically.

Conectiva estimated at least 500,000 users in Brazil alone (as of last November), with an expectation of one million in Latin America by early 2000; that's a lot of users! Just think of what one million more users will mean, in all areas from coding to kernel hacking, debugging to support, evangelizing and even just adding to that “network externality” effect. With Portuguese and Spanish as the first languages of over 500 million people, one million users is just the beginning.

Conectiva's translations have been thorough, compared to typical commercial software which is only half-heartedly translated (witness, for example, the Microsoft attempts at invading Asia; in particular, the Korea fiasco). In the past, typical translations from proprietary houses have had buggy character sets and ended up with a weird mix of English and whichever language. You'd pop up a menu, and some of the entries would be translated, but some not; every time something unexpected happened, the machine would lapse back into English. With Conectiva, you don't have these problems. Pop up any weird man page, contrive your machine to spit out error messages, and you'll get them in Portuguese. Together with the translation efforts of the KDE and GNOME projects, you've got double coverage. And should something untranslated come up, I'm sure the “open-source” approach to software building also works just fine for translation.

What Exactly is Conectiva Linux?

Conectiva Linux is a company that concentrates on translating Linux into Portuguese and Spanish. Its distribution, as one might infer, is called Conectiva Linux. Translation is the primary concern, so rather than creating the universe and assembling a distribution from scratch, Conectiva based its wares on Red Hat and worked from there. It's a reasonable decision, and Conectiva went its own way right from the start—it's definitely not a hacked Red Hat. While the typical hacker might have preferred a Debian base, that's all in the past and Conectiva is what it is now. Conectiva offers a standard edition and a server edition in either Portuguese or Spanish, although many packages bundle various oddities such as mouse pads, pins, pens, shirts, stickers, books and the like. Honestly, I don't understand why distributors like to have so many different packages, one package with different installation options might be

easier. Still, since Conectiva sees fit to divide Linux between server and standard, let's have a look at what each edition includes.

The standard edition is what you'd expect from a Linux distribution, translated into Portuguese and Spanish. A normal installation proceeds via the Red Hat installer, and even though during one install I got a couple of warning messages about RPMs, everything went along just fine, and on reboot, up popped the K display manager (KDM).

Conectiva chose KDE, GNOME, WindowMaker, MWM, FVWM and Icewm. Of these, FVWM, MWM and Icewm are about half translated, but the system has also been translated, so one isn't relying on the GUI for translation. These aren't just “alternatives” to KDE for those who won't conform. Every window manager actually works. KDE for all its perks does not look nearly sinister enough, and it's important to appreciate the variety of excellent window managers out there. Conectiva advertisements usually show funky WindowMaker or Enlightenment screenshots, and if one has an artistic flair, one might as well express it in all areas of life, including Linux.

Conectiva left off an entry GNOME/Enlightenment on the K display manager (accidentally, I presume). Therefore, if you want your life to look like a scene from The Matrix, you'll have to add an entry to KDM or start up GNOME by way of **telinit 3; gnome &** as root (oh no, not the GNOME warning message!). There is a general feeling that GNOME is the future (or at least has a very big future), especially considering Helix GNOME and even the Eazle project, so it would be nice if GNOME/Enlightenment were more highly featured. I look forward to the GNOME/Sawmill pairing, which I hope everyone including Conectiva picks up on, since it runs quite efficiently on minimal hardware. At the very least, GNOME's icons are outstanding.

Eu soy um Servidor

The server edition is a bigger installation in Portuguese and Spanish. “What's a distribution—a collection of software I can download for free on the Internet?” Caldera's Ransom Love once asked, and honestly, it's worth asking. Basically, that's exactly what a distribution is, with an installer, files placed into various directories and a bunch of configuration files all filled out. The server edition is packed with software that you can download over the Net, but it's probably most convenient for people who might not have high-bandwith connections. The documentation will walk you through complete networking server setup, configuration and administration.

Currently, the server distribution's number-one advantage comes in the form of four large manuals: Lars Wirzenius' *Linux System Administrator's Guide*, Olaf

Kirch's *Linux Network Administrators Guide*, and Conectiva's *Linux Server Guide* and *System Installation Guide*. Altogether, 1708 pages of documentation, and I must say, quite a good collection of work. These manuals cover everything server-related, they're thorough, and that's probably why they're included in the Edi<\#231><\#227>o Servidor. They're not condescending, and would be nice for computer-proficient Linux newbies, but even complete neophytes should find these books make for rapid learning and valuable reference. Most normal users would probably choose the standard edition of Conectiva, so they'll miss out on these excellent texts unless Conectiva decides to include them next time around.

The Meaning of it, Mostly

The point of Conectiva is not that it's the be-all, end-all, fastest, best-configured Linux with all the software squeezed magically on a 1.44MB floppy, a sentient living HAL 2000 that will make tea and beat you at chess. The point of Conectiva is that it makes Linux more accessible for hundreds of millions of potential users, and it works just fine. I've been coding on it, networking, configuring and customizing, and everything else your typical user does, for several weeks now and everything works. RPM installations, tgz installations, everything works fine without a single glitch. The libraries are current and in the right places, **ldconfig** knows where to find everything, and only a couple of libraries needed to be updated specifically for what I was doing-->plug-and-program, an improvement on plug-and-play. Setting up networking meant editing the same files as on Red Hat or most anywhere else, and there is a supply of the standard commercial applications, such as Netscape and StarOffice. Conectiva is very easy to use, and I just cannot find anything particularly wrong with it. If Linux is going to be championed into new territories, it should be presented as well as possible. Conectiva in many ways does this better than many stateside distributors, from flashy ads and cool T-shirts to giving users a choice of many well-configured desktops, with easy installation and configuration, excellent manuals, support, and an enthusiastic focal point for the Portuguese and Spanish speaking Linux scene.

There are a couple of bugs to be ironed out, for example the occasional warnings during installation, the absence of GNOME from the KDM, the weird entries on the KDM login (**gdm**, **postgress** and **xfs**), and root's mailbox getting flooded with Radius error messages. SVGAlib needs to be updated, SDL is missing, and if you program, you'll want to get the newest versions of your favorite libraries. Since 4.0 came out a while back there is now much to update for the next release. However, with so much translation work already done, Conectiva will be more able to concentrate on polishing and maintaining the software.

There was a time when distributions had such different libraries and files in different places that certain software would work on some distributions but not on others. These days, we don't really have that problem. Distributions are kind of a fetish; we like the name Red Hat if we like Heinz ketchup, and if we instinctively avoid brand names we instinctively avoid Red Hat, but it doesn't matter half as much as it once did. Sure, there are different config files, but you can change those without suffering. And maybe some libraries are old and need to be replaced, but that's part of running Linux; you're always out of date (and perhaps without one).

The other part of running Linux is that it always works, whatever name is on the box. I suppose, then, that this isn't a review as much as an announcement of good news. Well, thank you to Conectiva and everyone who has helped in the translation process for making that possible.

More Good News

Conectiva is about to release its latest version, Conectiva Linux 5.0, and has just shipped the latest episode of its monthly magazine *Revista do Linux*. Periodicals, though they be murder on trees, can unify communities.

The Good/The Bad

Jason Kroll (info@linuxjournal.com) is technical editor of *Linux Journal*. Even though he's a tad lazy, he tries to be an enthusiastic GNU/Linux cheerleader.

Archive Index Issue Table of Contents

Advanced search

Raritan CompuSwitch

Alex Heizer

Issue #73, May 2000

When I first unpacked the Raritan CompuSwitch, my first thought was, “This is TINY!”

- Manufacturer: Raritan
- E-mail: sales@raritan.com
- URL: http://www.raritan.com/
- Price: $495 US
- Reviewer: Alex Heizer

When I first unpacked the Raritan CompuSwitch, my first thought was, “This is TINY!” I was prepared for something a bit larger, being used to larger cases for simple A/B switches.

However, its small size made it a welcome addition to my office environment, and its power and flexibility was more than I expected from a unit dwarfed by a single-space rack-mount case. The CompuSwitch is an attractive machine in the endless sea of beige computer hardware. A black CompuSwitch would get more points from me on the coolness meter. When I was contacted by Michael Slaven, the OEM manager at Raritan, I was excited about the possible uses for a KVM (keyboard/video/mouse) switch in a Linux environment that worked well not only for Linux, but also for DOS, Macintosh systems and various strains of Windows. For people who use multiple Windows machines, or in a mixed environment consisting of DOS or Mac OS and some version of UNIX, one sure way to keep enough room on a desk for actual work is to use one monitor, one keyboard and one mouse to do all your stuff on all your computers. The easy, painless and relatively inexpensive way to accomplish this is with the CompuSwitch.

The Tests

I thought I would start my tests using a simple setup, in case I encountered any problems. For my controller setup, I chose an IBM AT keyboard (an original one with the great clicky keys, and the function keys on the left) with a Logitech PS/2 mouse and a Packard Bell 2160 monitor. See “Test Hardware” for a complete list.

Using a Windows 3.1 machine and one running straight MS-DOS would ensure as few variables in the mix as possible, while still enabling me to cover all three: keyboard, video and mouse. One machine used a PS/2 keyboard and mouse, the other required an AT keyboard and serial mouse. A simple plug-in, using their great cables made expressly for the switch, had me ready to hit the “go” button in less time than it took me to unpack it. Both machines came up with keyboard error messages on startup. A quick change to a no-name PS/2 keyboard put me back on track. Within a minute, I was happily computing away on two “obsolete” systems, switching effortlessly between them with the press of a button. Whether it was in the middle of serious graphics rendering, number crunching, or sitting idle while contemplating its digital belly button, each OS behaved as though it was being accessed by its own keyboard and mouse, displaying on its own monitor.

Windows 95 worked just as well, switching between it and DOS.

Tests using Caldera's OpenLinux 2.2 and Red Hat 6.0 both performed flawlessly, each having been configured for the test monitor. Neither KDE nor GNOME seemed to notice the unorthodox hardware configuration, and performance didn't suffer a bit.

Slackware was run first in console mode, and it recognized the monitor and keyboard at boot time. Running FVWM95 went perfectly too, with a normal setup.

Next, I hooked up a notebook which ran Windows 98. The PS/2 and USB ports presented no problems for the keyboard and mouse, and Windows recognized the external monitor with no problems.

Using a client's Novell and Windows NT servers also presented zero problems. Both recognized the PS/2 mouse and keyboard, even though they had been hot-swapped.

The only Mac I had to test with was an iMac, which ran both OS 8.6 and Yellow Dog Linux. With Yellow Dog Linux, I was unable to get the switch to work. I believe it was because this version of YDL didn't support USB very well, since it was brand-new when I got it. I had trouble getting even the original iMac

keyboard and mouse to work. The switch, however, worked with the Mac#OS. There was no USB-to-video adapter, so I was able to use only the keyboard and mouse, but I've seen setups that could benefit from even this modest hardware reconfiguration.

Switching between Linuxes, and between Linux, Mac and Microsoft operating systems, was as easy as anyone could possibly make it. By the end of the day, it was apparent that anyone who needed direct control of multiple physical machines running Windows, Novell, Mac or Linux would be able to do so without having to worry about where to place all needed hardware.

In every test, the CompuSwitch made it as easy to monitor and control each computer as if each were connected to its own keyboard, mouse and monitor. I even found myself just switching back and forth between my own Linux machines instead of mounting file systems, or using TELNET or FTP.

I would recommend the CompuSwitch (and already have) to anyone who has more than one computer to control. Graphic design service bureaus, network server rooms and home-based businesses that use multiple computers are environments that can benefit from using this switch. The cost of the unit is less than you would spend on the monitors, keyboards and mice it replaces, and certainly takes up less desk space.

Test Hardware

Alex Heizer (alex@concentric.net) is a network consultant who used to spend his time trying to develop ALUX, a Linux port for the TRS-80 Color Computer that is not free, but instead you get paid to use it. Upon realising this was unrealistic and unprofitable, he has turned his attention to Linux advocacy and education. He currently advocates in New Jersey, but is trying to get out.

Archive Index Issue Table of Contents

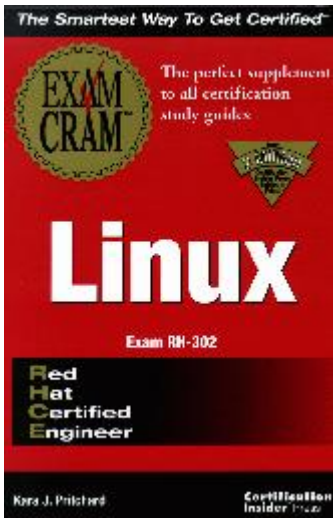
Advanced search

Linux Red Hat Certified Engineer Exam Cram

Andrew G. Feinberg

Issue #73, May 2000

The Red Hat Certified Engineer program (RHCE) exam has a new book to go with it, Kara J. Pritchard's RHCE Exam Cram.



- Author: Kara J. Pritchard
- Publisher: Certification Insider Press/Coriolis
- URL: http://www.certificationinsider.com/
- Price: $29.99 US
- ISBN: 1-57610-487-7
- Reviewer: Andrew G. Feinberg

Every time I walk into a bookstore, I see rows upon rows of Linux books and Windows books, all alike. However, the one area where Linux is lacking is in certification. Like myself, you may not believe that certifications are a true measure of ability. Nonetheless, Linux does have one certification exam in the form of the Red Hat Certified Engineer program (RHCE). This exam even has a new book to go with it, Kara J. Pritchard's RHCE Exam Cram.

The author assumes the reader knows a bit about UNIX in general and does not go into detail regarding basic skills. Chapter 1 is an introduction to the RHCE-300 exam itself, and Chapter 2 is an introduction to the history of Linux and free/open source software. The book includes a self-assessment at the beginning to determine whether one needs it. If you want a tutorial, this is not for you. For UNIX users who may not be administrators, Ms. Pritchard gives a good introduction to basic administration concepts in Chapter 3. Chapter 4 goes into great detail about the basics of TCP/IP and related services. Chapter 5 covers the basics of a Red Hat Linux installation, with which some readers might not be entirely comfortable, and after a brief interlude of basic system administration, Chapter 6 immerses the reader in dual-boots, kickstarts and other wonderful features of the Red Hat Linux installer. Chapter 8 covers quotas, run levels and the nitty-gritty of custom kernel compiles, as well as teaching one how to deal with crontab and profile files.

Chapter 9 does a wonderful job of showing how to configure the X Window System from scratch, as well as clueing a user in on several popular window managers including the GNOME and KDE environments. Chapter 10 describes how to configure Apache, Sendmail, FTP, DNS, NFS and Samba, as well as DHCP, Time Configuration and Squid. I found this section to be the most informative and useful outside of the realm of Red Hat Linux. I consulted the Samba section of the chapter when I had to help a friend configure a home network with Samba, which I had never done before.

Chapter 11 shows how to take a newly networked system and secure it, something every Linux administrator should know, including the basics of NIS, working with PAM, tcp_wrappers, routing and firewalls with ipchains.

Chapters 12 and 13 explain the process of the installation exam and the debugging exam, with Chapter 14 being a practice test and Chapter 15 its answer key. I am happy to report that I scored a 47 out of 50 on my first try, thanks to Ms. Pritchard's wonderful tutorials and test hints.

This being an exam preparation book, Ms. Pritchard does a good job of placing sample questions pertinent to each section throughout the book at the end of each chapter. She also lists external resources, such as HOWTOs, FAQs and other books on the subject at the end of each chapter. Every chapter has numerous exam hints, tips and tricks that are useful when you take your test.

Linux certification might not be the best way to ensure quality and skill of administrators, but it is a step toward getting Linux accepted in the corporate marketplace. Certification programs may produce cookie-cutter administrators, yet they will have very marketable and consistent skills useful not only with Linux, but with UNIX in general. Ms. Pritchard's book does a great job of

introducing someone to, and preparing them for, the Red Hat Certified Engineer exam, and hopefully, will lead Linux users to bigger and more interesting learning experiences.

Andrew G. Feinberg is a developer of Debian GNU/Linux. In his spare time, he can be found working on the High School Linux User Group project (hs-lug.tux.org) or attending classes at Walt Whitman High School in Bethesda, MD. He can be reached with questions, comments, or letters of college acceptance at andrew@ultraviolet.org.

Archive Index Issue Table of Contents

Advanced search

JavaScript Application Cookbook

Ralph Krause

Issue #73, May 2000

A good resource for anyone who wants to take advantage of the power of JavaScript.

- Author: Jerry Bradenbaugh
- Publisher: O'Reilly & Associates
- E-mail: info@oreilly.com
- URL: http://www.ora.com/
- Price: $34.95 US
- ISBN: 1-56592-577-7
- Reviewer: Ralph Krause

The *JavaScript Application Cookbook* written by Jerry Bradenbaugh and published by O'Reilly is a good resource for anyone who wants to take advantage of the power of JavaScript. Instead of providing a language tutorial or a collection of routines, the book contains fully functional JavaScript applications. Some of the applications included are a client-side search engine, a shopping cart, a context-sensitive help system, and a drag-and-drop application that lets users build their own DHTML e-mail greeting cards. The applications are fully functional and show how powerful JavaScript can be.

The application source code and images are available from O'Reilly's web site as one big Zip file. The book does contain the source code listings, but since some of them contain over 400 lines, I suggest downloading the file instead of trying to type the code from the book. While the code is not heavily commented, Mr. Bradenbaugh thoroughly explains it in the body of the book.

I downloaded the source code and ran all of the applications first on my home computer and then over the Web from my home page. All the applications run quickly within a browser and most of them load fairly quickly, even over a dial-up connection.

The book is made up of eleven chapters, each (except for one) containing one application. The chapter that does not include an application contains JavaScript routines that can be added to your existing code. Quite a few of the routines presented in this chapter are used in the other applications in the book.

All the chapters are laid out in the same way. The application features and JavaScript techniques used are stated at the beginning of the chapter, along with an overview of the application and any special browser requirements. Screen shots of the running application are also provided.

Next comes a syntax breakdown which contains a source code listing, followed by an explanation of how each section of the code works. These breakdowns are not JavaScript tutorials; instead, they show how the code interacts with the browser and explain some of the more powerful and exotic JavaScript features used by the application. Mr. Bradenbaugh's explanations are clear and concise, and he includes quite a few sidebars and web addresses for more information on complex subjects.

Each chapter ends with a section called Potential Extensions. These sections provide thoughts on how the reader can extend and modify the application that was just presented. While some code snippets are provided, the majority of the work is left to the reader to accomplish.

The book contains three appendices. The first is a JavaScript reference. The second appendix provides links to JavaScript- and web-related resources, as well as sites with applications similar to the ones presented in the book. The third appendix contains a Perl/CGI overview.

Mr. Bradenbaugh provides quite a bit of explanation and code showing how to have your code determine if it is running on a Netscape or a Microsoft browser, then run correctly when doing such things as DHTML. I tried running some of

the applications on both Netscape and Microsoft browsers, and they worked correctly each time.

The book's applications demonstrate good attention to detail. Such things as positioning buttons correctly when creating a page, positioning new browser windows so that they don't completely cover existing windows, and putting search results in alphabetical order are hallmarks of polished applications.

I did not find any errors in the book, and there were no errata on O'Reilly's web site when I looked. The only thing that was not correct as stated in the book are the web addresses for Internet Explorer information. These pages were moved by Microsoft when they redesigned their site.

I found the *JavaScript Application Cookbook* to be a useful book, containing applications that can be used right away. The explanations and code breakdown do a very good job of illustrating the power of JavaScript. While not a JavaScript primer, the book will be useful for someone with JavaScript experience who wants to create powerful programs.

Ralph Krause is a freelance computer consultant in Michigan. His current career goal is to stay out of automotive paint shops. He can be reached via e-mail at rkrause@netperson.net.

Archive Index Issue Table of Contents

Advanced search

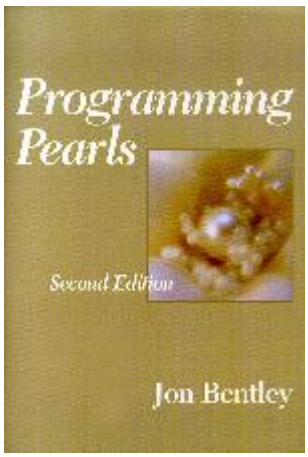
Programming Pearls, Second Edition

Harvey Friedman

Issue #73, May 2000

This book is not about programming in Perl, even though the Perl language is referenced twice.

- Author: Jon Bentley
- Publisher: ACM Press / Addison Wesley
- E-mail: ACMHELP@ACM.org
- URL: http://www.acm.org/
- Price: $24.95 US
- ISBN: 0-201-65788-0
- Reviewer: Harvey Friedman



To head off any misconceptions our spelling-challenged readers might have after seeing the title, this book is not about programming in Perl, even though the Perl language is referenced twice. It is a revision of a book published in 1986 that comprised an edited selection of columns Bentley wrote for the journal *Communications of ACM*. ACM stands for the Association for Computing Machinery, an organization of and for hardware and software professionals and students, formed back before computing technology was as

specialized as it is today. As a member in the late 60's and early 70's, I eagerly read Bentley's columns, even though at times I became frustrated at not seeing the solutions he offered until after reading the hints. His first edition of *Programming Pearls* became an instant classic with examples from the languages C and FORTRAN, but with pseudo-code for the algorithms and solutions so that one could implement in one's favorite language. I used some of the ideas from the book when designing and implementing application access software for relatively large atmospheric science binary data files from a CD-ROM data plate.

Since Bentley works at Bell Labs, he has had the opportunity to consult on many interesting problems, the more interesting of which he uses as examples in this book. There are 15 chapters which he refers to as “columns” (because originally, each was based on a column from *CACM*). Each includes a discussion of the problem he considers, principles he believes one can derive from that column, and problems for solution and/or further thought. At the end of the book are hints (further questions providing insightful ways of approaching a solution) corresponding to each column's problem set, and outlines of solutions in a section following the hints section. Each column also has a section called “Further Reading”, where he lists books or journal articles related to the column. It was a minor disappointment that there was not a collected bibliography at the end of the book, where all the further reading material was properly referenced with ISBNs where possible.

Finally, some of the chapters have sidebars at the ends; these are well-written narratives with interesting case studies or anecdotes. I was particularly taken by sidebar 13.8, where he presents Doug McIlroy's spelling checker. Fitting a dictionary list of 75,000 English words in the 64KB address space of a PDP-11 computer sounds impossible. Even having Bentley explain the trail of space-compressing techniques used and seeing timings for the 20-year-old “spell” doesn't take away the sense of awe.

To bring the second edition up to date, Bentley uses only C and C++ example programs for his pseudo-code. He rewrote programs for all the pseudo-code in the book, ran them on his 400MHz Pentium II with 128MB of RAM running MS Windows NT 4.0 and presented the timings in the appropriate columns. I would think if he really wanted to be up to date, he would run them on Linux. That might be a good exercise for someone playing with a Linux system who has time on their hands: download Bentley's C and C++ programs, to see how the times compare when run on a similar hardware setup. In Appendix 1, “A Catalog of Algorithms”, Bentley provides college instructors of algorithm courses with ways of presenting the material in his book. Sorting, searching, string algorithms, vector and matrix algorithms, generating pseudo-random integers and numerical algorithms all have one or more sections.

Appendix 2 is a brief, back-of-the-envelope calculation quiz to evaluate one's proficiency at making reasonable numeric guesses. After trying it, I need more practice. Appendix 3 has cost models of time and space; for computers, naturally. Appendix 4, Rules for Code Tuning, comes from his 1982 book *Writing Efficient Programs* and lists 27 rules. The main categories are space-for-time rules, time-for-space rules, loop rules, logic rules, procedure rules and expression rules. He lists them, then references the section of the book which has examples. Appendix 5, C++ Classes for Searching, is just what it says it is.

This second edition has lost nothing in the rewriting. I only hope a new generation of programmers can appreciate it as much as those of us who learned our approaches to solving programming problems by reading the classic first edition.

Harvey Friedman can be reached via e-mail at fnharvey@u.washington.edu.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Bourne Shell Scripts

Randy Parker

Issue #73, May 2000

Scripting with EX and Here files.

At some point in your dealings with UNIX and Linux, you may need to write a script which edits critical system files such as /etc/passwd or the rc boot scripts. Whatever the file, if your script messes it up, your machine will suffer because of it. Half-hearted scripting, which may be sufficient for normal files, will not work in these situations.

Consider the root password. If you need to modify the root password on one machine, it's quite easy using the **passwd** command. But if you need to change the root password on 50 or more machines, the problem is much more difficult. It could still be done by hand, but this would be time-consuming and prone to errors. One way to solve this problem is by using EX.

If you view the man page for EX on Red Hat Linux, you will get the man page for **vim** instead. This is no mistake. EX is the engine underneath the familiar **vi** interface. All the power of vi is available from within your scripts, if you know how to access it.

Let's start with some simple edits and work our way up to the root password change example.

Listing 1

Listing 1 shows a simple multi-line file. Suppose we wanted to remove the first line containing the word “three”. We could write the short script in Listing 2 to accomplish this easily.

Lines 1 through 6 in Listing 2 should be familiar to anyone who has done shell scripting. If you have never written a script, I recommend the book *Learning the bash Shell* (see Resources) to get started.

Listing 2

Line 5 is where the interesting stuff begins. We call EX with the -s option to tell EX to run in silent mode. The file we are operating on is entered next, in the **$TARGET** variable. The uninteresting output of EX is redirected to /dev/null because we don't want it to clutter any output of the script itself. You can delete **>/dev/null** if you would like to see this information during debug. The **<<E_O_F** marks the beginning of a “Here” file.

A Here file is a way of placing the contents of a file within a script. The line after **<<E_O_F** is the first line of the file. The Here file consists of all lines up to the line beginning with E_O_F. This tag (which ends the Here file) can be any unique character combination. We are using E_O_F here out of convention. The “<<” has the effect of feeding the contents of the Here file to EX, which uses those commands to edit the **$TARGET** file. Notice that the Here file is not indented. If we were to change **<<E_O_F** to **<<-E_O_F**, adding the “-”, we could indent the Here file with **TAB** characters. If those **TAB** characters somehow became spaces, perhaps by a copy-and-paste operation, the Here file will cease to work, the E_O_F will never be seen and the contents of your script below the E_O_F marker will be fed to EX, creating unpredictable output. For this reason, I make it a personal policy to never indent Here files. For more information on Here files, see the book *UNIX Power Tools* (in Resources).

The first line of the Here file is line 6, which contains a 1. This forces the line pointer in EX to the first line of the file. While this is not completely necessary for the script to work, it does eliminate any confusion as to where subsequent searches begin during your edit. I find it to be a helpful programming practice.

Line 7 is a regular expression. EX uses this regular expression to search through the **$TARGET** file and set its line pointer to the first line containing the word “three”. In listing 1, this is line 3. The trailing “d” in line 10's regular expression tells EX to delete that line. Line 8 is familiar to vi users. It says to write, quit and “don't argue with me”. Line 9 is the E_O_F tag, which marks the end of the Here file.

Listing 3

We could add lines to the file with the **a** or **i** commands. Listing 3 shows examples of this. Notice the trailing “d” is gone from the regular expression. We are using the regular expression to set the line pointer, not delete the line. The **a** appends a line after the line defined in the regular expression, and the **i** command inserts a line before the line defined in the regular expression. The **.** command tells EX to leave edit mode, similar to pressing the **ESC** key in vi. The last EX example in Listing 3 changes the 1 to a $, forcing the line pointer to the

end of the file. We then append a line, which has the same effect as this command:

```
echo "This line is added at the bottom of the \
file." >> $TARGET
```

Listing 4

Now that we have simple edits under our belt, let's look at the more complicated problem mentioned earlier of changing multiple root passwords. Once we have changed one password with the passwd command, we can copy and paste the password hash from /etc/shadow into a variable in a script. Listing 4 contains the script. The **REPL_STRING** variable contains the new password hash. Notice we had to escape "$", "." and "/" with backslashes to prevent EX from interpreting these as special characters and giving unpredictable output. As you see, the only major difference between this script and the earlier examples is the regular expression.

Due to space constraints, I will not fully explain the regular expression here except to say it has the effect of "remembering" everything on each side of the existing password field and adding that to each side of the **REPL_STRING** variable, making up a new root line in the /etc/passwd file. At the end of the script, we use **pwconv** to put the new password hash into the /etc/shadow file. This script can be put in a place that all machines can mount and can be remotely executed by each machine, giving them all the same root password.

As you can see, the file editing power of your scripts with EX is limited only by your knowledge of regular expressions. If you don't currently use regular expressions, they are well worth taking the time to learn. For more information on regular expressions, read the book *Mastering Regular Expressions* (see Resources).

Resources

Randy Parker has been using UNIX since 1993 and Linux since 1995 (Thanks, Lee!). When not bashing his head flat against a computer screen, he enjoys playing guitar and getting his brother's boat dirty. He can be reached at rap@dfw.nostrum.com.

Archive Index Issue Table of Contents

Advanced search

xv: the X Viewer

Marjorie Richardson

Issue #73, May 2000

When you want to take a quick look at a graphics file, XV is the application to use.

Most of the graphics printed in this magazine come to us electronically, and the first way we look at them is with **xv**, an interactive viewer for the X Window System. This program was written by John Bradley and has many pages of documentation. The man page tells you only the name of the documentation file, namely xvdocs.ps; you have to find out where it was installed (perhaps /usr/local/lib/). The program is quite simple and intuitive to use, and you can figure out its basic uses whether you read the manual or not.

xv will display just about any type of graphics file you happen to have: GIF, JPEG, TIFF, PBM, PGM, PPM, X11 bitmap, BMP and many more. It will even display PostScript by calling up Ghostview to help it out. It is available by anonymous download at ftp://ftp.cis.upenn.edu/pub/xv/ and has a home page at http://www.trilon.com/xv/. **xv** is a shareware program that is free for personal use. There is a $25 US registration fee, and site licenses are available.

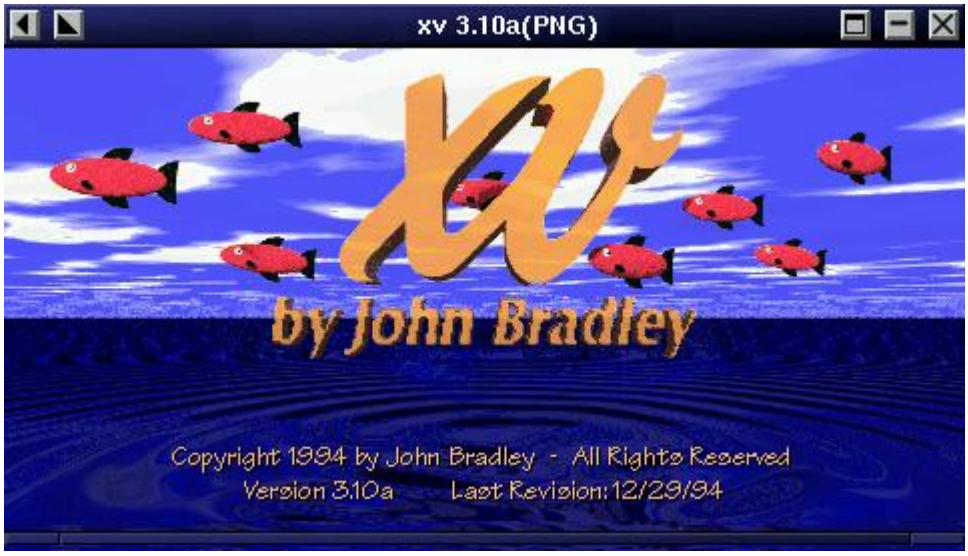


Figure 1. Main Window

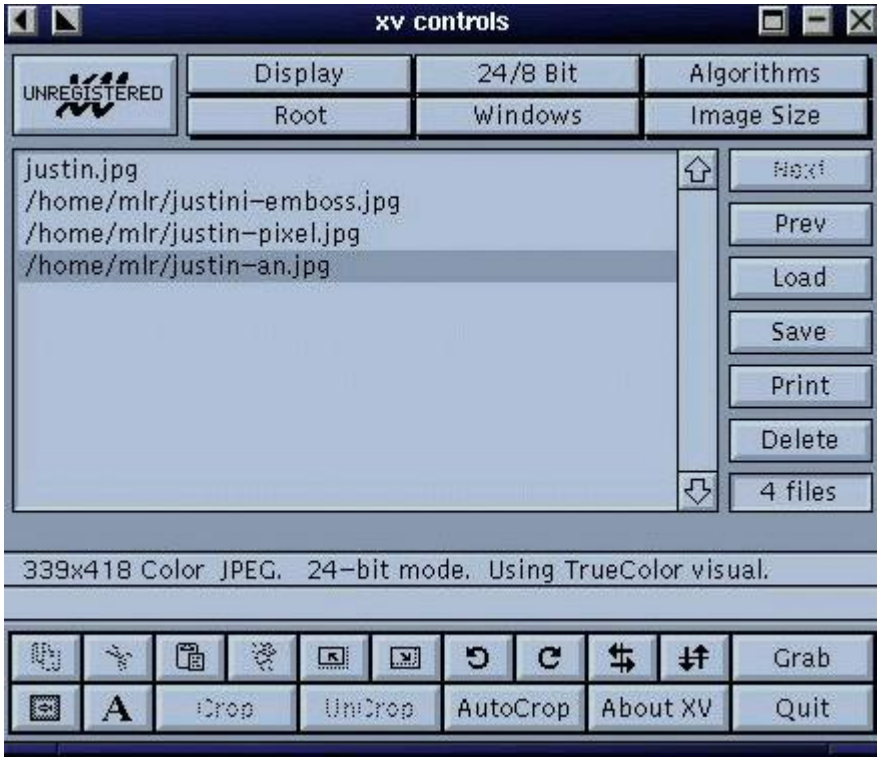


Figure 2. Control Window

Typing **xv** alone at the command-line prompt brings up a window containing the xv title page with bright red fish swimming through the air (see Figure 1). Clicking the right mouse button in the window brings up the control window (see Figure 2), while pressing the letter **q** quits the program.

Figure 3. Justin

To load a picture, click on “Load”, and a window will come up showing the contents of the current directory. Double-click the image file you want to view, and that image replaces the red fish in the main window. Alternatively, you can type the file name on the command line:

```
xv 3964f3.jpg
```

This command displays the picture of my youngest grandchild, Justin, shown in Figure 3. Wild cards are recognized, so typing **xv *.gif** will load every GIF file in the current directory, displaying one after another each time you press the space bar.

Looking at Figure 2 again, you'll see that there is a double row of buttons across the top of the control window. Clicking on any of these buttons causes a menu to drop down, showing various options. Options not available will be grayed out. For example, if I click on “Display” while viewing Justin's picture, I am given the options of “raw” and “smooth”, but “dithered” is grayed out. Also, “Use standard colormap” is checked and all other options are grayed out.

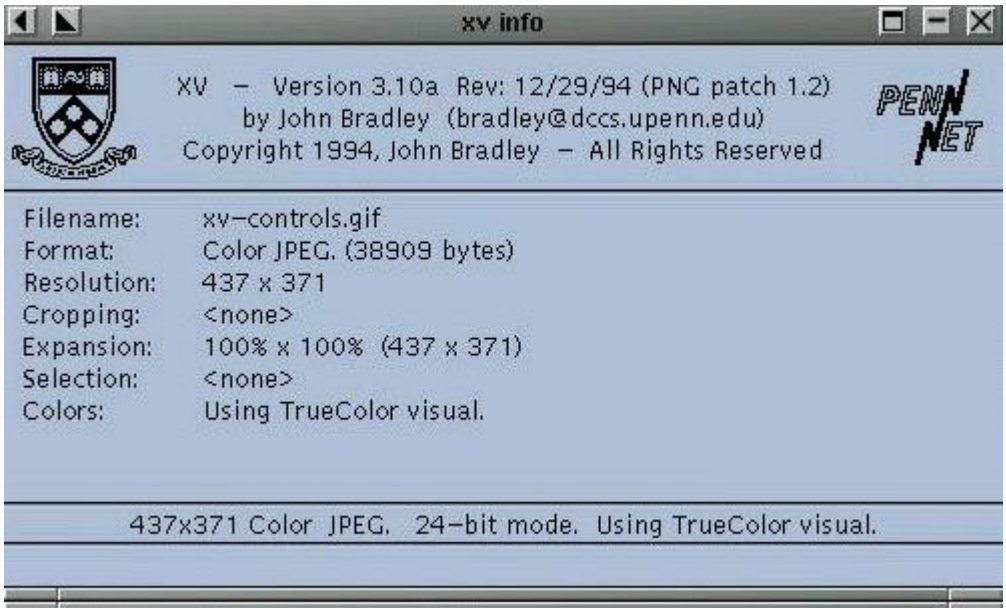


Figure 4. Image Info Window

In the menu obtained by clicking on “Windows”, you will find “About XV”, which you tells you about registration fees, and “XV Keyboard Help”, which tells you how the program will react to certain key and mouse presses. The most useful options in this window are “Color Editor” and “Image Info”, the latter in particular. I use the Info option for every image I display to determine if the picture is high enough resolution to print well. The image information for Justin's picture is shown in Figure 4. Looking at this information, we see that the image is in JPEG format and has a resolution of 339x418. This will give us a picture that would print reasonably well as an author photo. It's a bit grainy at the larger size we are showing here. Any smaller, and our Associate Editor Darcy Whitman would be writing for a new picture. The resolution is also shown toward the bottom of the control window.

The menus for “Image Size” and “24/8 Bit” are fairly self-explanatory. Image size is nice in that it gives you many choices for resizing the picture, including setting it yourself or as a function of percentages of current size or pixels. The “Root” menu I leave alone—the name is enough to deter me from using it. The “Algorithms” menu can provide some fun, although not as much as you might have with programs such as the GIMP. Figure 5 shows an embossed image of Justin, while Figure 6 is a pixelized version. The oil painting option was not pretty—it looked like sections had been scraped off with a knife.

Figure 5. Embossed Image

Figure 6. Pixelized Image

The buttons down the right side of the control window will allow you to “Save”, “Print” or “Delete”, as well as load images. Delete will let you delete from the viewing list, or from the list and the disk both.

Rows of buttons along the bottom allow you to rotate the image 90 degrees (right or left) or turn it upside down, if you wish. Play with them—it's fun for a minute or two. The “A” button gives you the ability to annotate the picture as I did for Figure 3, which can be quite useful in remembering when a photo was taken. I use the “Crop” button quite often; in fact, this picture of Justin is cropped from one in which I was holding him (see Figure 7). Clicking and holding down the left mouse button as you drag it in the window draws a frame which defines the area to be cropped.

Figure 7. Uncropped Picture

I also use the “Grab” button quite frequently in order to make screen shots. Click on this button, set a delay time (e.g., 1 second), click on “Autograb”, put your cursor in any window, and that window will appear in XV's main window. Or you can use “Grab” and define the area you wish to view by holding down and dragging the left mouse button.

XV is truly a useful program and one that I use on a daily basis. Try it; I think you'll like it.

Marjorie Richardson is Editor in Chief of *Linux Journal*, the best job in the universe. She loves all her grandchildren equally, even though she used only Justin's picture in this article.

LINUX JOURNAL

Linux Use Rocketing at Jet Propulsion Laboratories

Drew Robb

Joe Zwers

Issue #73, May 2000

A look at how JPL scientists are using Linux to build better spacecraft and make accurate calculations.

Few technologies are more exacting than that of designing and running space probes. A trajectory error as small as one inch per mile, for instance, can result in missing the target by tens of thousands of miles. Now add stringent budgetary constraints, and you have some idea of the daily working environment of the scientists and engineers at the Jet Propulsion Laboratories (JPL) in Pasadena, California.

Linux at NASA

With their long history of mainframe-based computing, NASA scientists were quick to spot the benefits of Linux. The NASA Goddard Space Flight Center, for example, uses Linux with its airborne Scanning Radar Altimeter to study hurricanes. NASA's Beowulf-class clustered computing project has also made use of Linux for several years. Further, one NASA employee maintains a site on the use of Linux for x-ray astronomers, and another is harnessing it for the on-board navigational computers of space probes. It's no surprise, therefore, that NASA's JPL is now utilizing Linux in two very different projects: one to build better spacecraft and the other to see that they arrive at the right place.

Building Better Spacecraft

When you are two billion miles from earth, you can't just pop out to a hardware store to get a spare part. JPL's Advanced Thermal and Structural Technology Group (ATSTG), therefore, is responsible for ensuring all space probe components can withstand the rigors of space travel. The ATSTG works on advanced technologies for thermal, cryogenic and structural space systems

including aerobots, rovers, mechanical coolers and heat pipes. They have recently integrated Linux into the business of building better spacecraft.

Thermal engineers play a key role in the design of space hardware. “Thermal engineering provides designs that maintain the required temperatures at specific space hardware sites through the range of all modes of operation and for the life of the hardware in space,” says Technical Group Leader Jose Rodriquez. “Temperature requirements for electronics, mechanisms, detectors and lasers are critical to mission success.”

The ATSTG uses a number of analysis tools to develop analytical models to carry out design trade studies. The best, cheapest and most complete integrated thermal analysis package available is the Thermal Synthesis System (TSS). It allows thermal engineers to create a thermal model of the hardware in a simulated space environment. They can then achieve a better understanding of how well the system will actually work when it is out in space. At the time it was developed, the TSS graphical interface requirements could be met with only high-end HP machines running UNIX. When the ATSTG wanted a version for its own use, the only free versions of the software available were for HP running UNIX or Linux on a PC. Given the choice between spending many thousands of dollars to buy or rent an HP, or spending $150 US to buy Red Hat Linux and Accelerated X, they opted for the Linux version of TSS. The group also installed Linux TSS on computers at two subcontractor facilities.

TCP/IP is used to transmit data and models back and forth between Linux and Windows computers. “Because of the high expense associated with the HP machines, it is necessary to keep them utilized constantly to generate enough resources to pay for the equipment,” states Rodriquez. “This is done by keeping as few machines as possible, which means that engineers can't easily access these computers during normal working hours and have to work early or late hours to get the work done.” Having Linux allows the ATSTG members to run TSS right on their own PC whenever they need it, eliminating costly downtime and enabling projects to be completed sooner.

The Right Place at the Right Time

Building a better spacecraft is fruitless, however, if it never arrives at its intended destination. While some people have trouble navigating their way across town, the scientists at JPL have to plot a trajectory that will allow a space probe to intercept a planet that is billions of miles away and traveling at 15,000 miles an hour. If you arrive a day late after a three-year journey, you've missed the target by hundreds of thousands of miles. As well as difficulties in hitting its space targets, there are also budgetary targets that NASA personnel are expected to meet. Linux is key to achieving both. “Smaller projects (DS1, Mars Pathfinder, etc.) are trying very hard to cut costs while achieving all scientific

goals," says Peter Breckheimer, Technical Group Supervisor in the Navigation and Flight Mechanics Section of JPL's Systems Division. "Computers to run the navigation system have been very expensive in the past due to the high level of performance required."

Breckheimer's group has the responsibility of designing, implementing delivering, and sustaining the Institutional Navigation System Software (INSS) in all flight projects (Galileo, Cassini, Mars, DS1, Stardust, etc.). Consisting of over 160 programs written primarily in Fortran 77 and C, Perl, Tcl/Tk and HTML, INSS contains 4.5 million lines of source code. INSS is designed to be readily portable to new platforms as they develop (particularly Linux).

When JPL switched from mainframe to VAX in the eighties, it required 15 man-years of software conversions. Later, for the move to HP-UX, engineers used only standard features within the Fortran 77 and C compilers. As a result, portability issues are easy to deal with. Going from HP-UX to Linux, therefore, took minimal work and even tools like Perl and Tcl/Tk worked correctly the first time. "There are tools we have experimented with to convert c-shell to NT scripts, but we haven't been successful yet," reports Breckheimer. "As for Linux, no conversion was necessary. The scripts worked properly with no changes."

Benchmarking Linux

Breckheimer has benchmarked a number of computers since 1984, using a program called **nbodyf**. This program takes an initial vector and integrates it with the equations of motion for all planets in the solar system over a five-year period. In 1984, a DEC VAX 11/780 took an hour and a half to run the program. Now an HP J5000 can do the same thing in under three seconds. While speed is important, the computer system must also get the correct result. Therefore, the answer is tested using a subset of INSS containing about 1 million lines of code. When run on the benchmark tests, the $2500 PC running Linux compared very favorably with results obtained on an HP 755 which cost over $30K. Linux Red Hat version 5.0/5.1 was installed on a generic 200MHz PC with 64MB RAM. It ran the nbodyf program in 14.5 seconds, versus 13.3 seconds for an HP 755/125 running HP-UX 9.07, and beat the times for the HP J21, HP C110 and HP 755/99. On the other tests, it ran as well or better than the 755/125. Later tests were run using a 450MHz PC, and its performance improved roughly in proportion to clock speed. "An additional side benefit we observed was a slight improvement in accuracy," says Breckheimer. "The Fortran 77 compiler, obtained from Absoft, utilizes 80-bit registers for some of the built-in functions used by our navigation software. On all previously tested UNIX platforms, we were limited to 64-bit arithmetic, so the Linux version of Absoft Fortran 77 offered some definite improvement."

The Future for Linux at JPL

Year after year, Linux is gaining greater acceptance within NASA. Given the huge investment that exists in older systems, it is not likely that Linux will replace Windows or UNIX in the short term, but Linux is working well where it has been used. “Having Linux's desktop functionality tremendously improves the engineers' efficiency and speed,” says Rodriquez. “Projects greatly benefit when engineers have readily available desktop analysis tools.” His one complaint, however, was the limited amount of peripheral hardware that Linux supports. But with so many computer vendors now offering Linux and the increasing number of Linux applications, this situation is rapidly changing. Breckheimer, who has been working with INSS through several platform incarnations, feels that Linux “opens up a significantly cheaper path to future computing for many of the projects we support. While HP, Sun and SGI workstations dominate our operations environment, Linux-based PCs may make a significant impact soon. As more projects accept Linux, computer costs for navigation systems will continue to decrease significantly.” So, while you don't have to be a rocket scientist to appreciate the advantages of Linux, these days even rocket scientists are beginning to adopt it as an alternative to Windows.

Drew Robb (drewrobb@mediaone.net) is a freelance writer from Tujunga, California, specializing in technology issues.

Joe Zwers is a freelance writer from Glendale, California, specializing in technology and law.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

The Linux Trace Toolkit

Karim Yaghmour

Michel Dagenais

Issue #73, May 2000

Analyzing performance is one of the most important tasks of a system administrator; here's how to do it using Linux.

As recent Linux history has shown (Mindcraft, anyone?), performance is not only good publicity, it's important. Yet current means of measuring performance offer only global statistics about the whole system or very precise data about an isolated application. Moreover, these often fail in helping the programmer or the system administrator to isolate a performance bottleneck resulting from the interaction of complex internetworking applications, which are more and more common. The Linux Trace Toolkit (LTT) addresses these issues and provides users with a unique view of the system's behavior with minimal performance overhead (< 2.5%).

LTT's Architecture

In order to be extendable and accomplish its task without hindering system performance, LTT is designed to be as modular as possible. In fact, it would be wrong to call it a “tool” since it is composed of many pieces that, grouped together, fulfill the desired function. This toolkit is implemented in four parts. First, there is a Linux kernel that enables events to be logged. Second, a Linux kernel module takes care of storing the events into its buffer and then signals the trace daemon when it reaches a certain threshold. The latter then reads the data from the module, which is visible from user space as a character device. Last, but certainly not least, the data decoder takes the raw trace data and puts it in a human-readable format while performing some basic and more advanced analysis. This decoder, as will be discussed further, serves as the toolkit's graphic and command-line front end.

Installing LTT

The LTT tar.gz archive can be found at http://www.opersys.com/LTT/ and contains the following items:

- Copying: the GNU GPL License
- Help: LTT's help files in an HTML-browsable format
- TraceDaemon: the directory containing the trace daemon
- TraceToolkit: the directory containing the trace toolkit front end
- patch-ltt-kernelversion-*yymmdd*: the kernel patch of *yymmdd* kernelversion
- **trace**: a script to start the trace daemon
- **tracecpuid**: another script to start the trace daemon
- **tracedump**: a script to dump the content of trace
- **traceanalyze**: a script to analyze a trace
- **traceview**: a script to view a trace in graphical form

The scripts are there to speed up the tools' most common usages, but the tools can be summoned directly without any script.

To install LTT, simply follow the instructions that come with the LTT package. The first and hardest step is patching the kernel. Once this is done, configure the kernel and compile it. Note that there is an option for compiling with or without the tracing code. When compiled without, the resulting kernel operates as if you hadn't applied any patch to it. Next, compile the trace daemon and the trace toolkit graphic front end and put them in your favorite directory (/usr/bin or /usr/local/bin for example). Reboot with the LTT patched kernel, and you're ready to go.

Sample Tracing Session

To demonstrate the toolkit's operation, we traced 10 seconds' worth of system operation. During those 10 seconds, two commands were issued: **dir** on a directory not accessed since system boot (i.e., not present in the dcache) and **bzip2** on a 10MB file. The system was booted in single-user mode (in order to have as few applications running as possible, and therefore isolate the operation of the observed applications) using the modified kernel. Note that no events are recorded by the kernel module until the trace daemon has issued the **start** command to it using the **ioctl** system call. The following command was issued to start the tracing:

```
trace 10 out140
```

trace is a script that takes two arguments: the number of seconds the trace should last and the base name for the output file. Two files are produced: out140.trace and out140.proc. The former holds the data recorded by the kernel module, and the latter, the content of /proc when the trace started. Using these two files, we know what the system looked like before we started tracing it and what happened during the trace. Hence, we can reconstruct the system's behavior.

Note that the trace daemon accepts many command-line options used to configure the kernel trace module. For instance, one can specify the events to be traced and the desired level of details. One can also specify whether CPU IDs should be recorded for SMP machines. Since LTT fetches the calling address for system calls, you can specify at which calling depth this address should be fetched or which address range it is a part of.

Viewing the Result of the Trace Session

This is the most interesting part about LTT and what differentiates it from other tools. Even though the current example is rather simplistic, more elaborate traces can be generated and analyzed just as easily. To view the trace in the graphical front end, simply type:

```
traceview out140
```

The **traceview** script accepts one argument: the base name of the generated trace. It then executes the trace toolkit front end, passing it the out140.trace and out140.proc files. Note that the front end can be invoked manually and can be used as a command-line tool as well. In the latter case, it can be used to dump the contents of trace in a human-readable format or perform a detailed trace analysis. The complete options are described in the documentation included with LTT.

A sample display of the front end can be seen in Figure 1. As usual, there is a menu, a toolbar and the data. To present the data, the tool uses three thumbnails. The first, Event Graph, presents the events in a graphical way. On the left, a list of all processes that existed during the trace is presented. On the right, a scrollable graph displays the transitions that marked the time frame being displayed. A vertical transition means the flow of control has been passed to another entity. The color of the vertical line used for the transitions indicates the type of event that caused the transition. Blue is for a system call, grey for a trap and white for an interrupt. Horizontal lines indicate the time spent executing code belongs to the entity in the process list at the same height. A green line is for application code and an orange line is for kernel code.

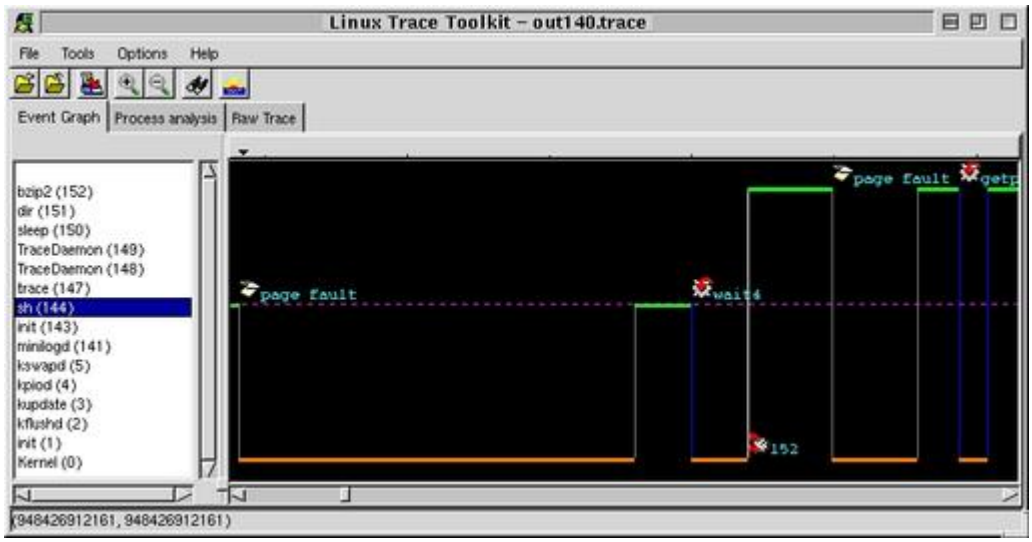


Figure 1. LTT Front End

In Figure 1, we can see that **sh** was running and a page fault occurred. Control was transfered to the kernel, which did some operations and then gave control back to sh. The latter executed for a while, then did a **wait4** system call. This, in turn, gave control back to the kernel. After a couple of operations, the kernel decided to do a task switch and handed control to process 152, bzip2. While bzip2 was running, another trap occurred. Toward the end of the trace, we can see bzip2 made a system call to **getpid**, and on returning, continued executing.

Figure 2 presents a more interesting case. For illustration purposes, the trace portion was zoomed out; some icons overlap, but the meaning is fairly straightforward. Here, bzip2 tries to read something, but the kernel doesn't find it. bzip2 starts waiting for I/O (input/output) and the kernel schedules the idle task. Two milliseconds later, the hard disk sends an interrupt to signal that it finished reading (that's the icon with a chip and an exclamation mark with a 14 beside it, 14 being the IDE disk IRQ). **bzip2** is then rescheduled, stops waiting for I/O and returns to its normal execution. Here, we can see exactly how much time bzip2 spent waiting for I/O and the exact chronology of events that led to it losing CPU control and regaining it. When trying to comprehend complex interactions between tasks, this type of information can make all the difference.

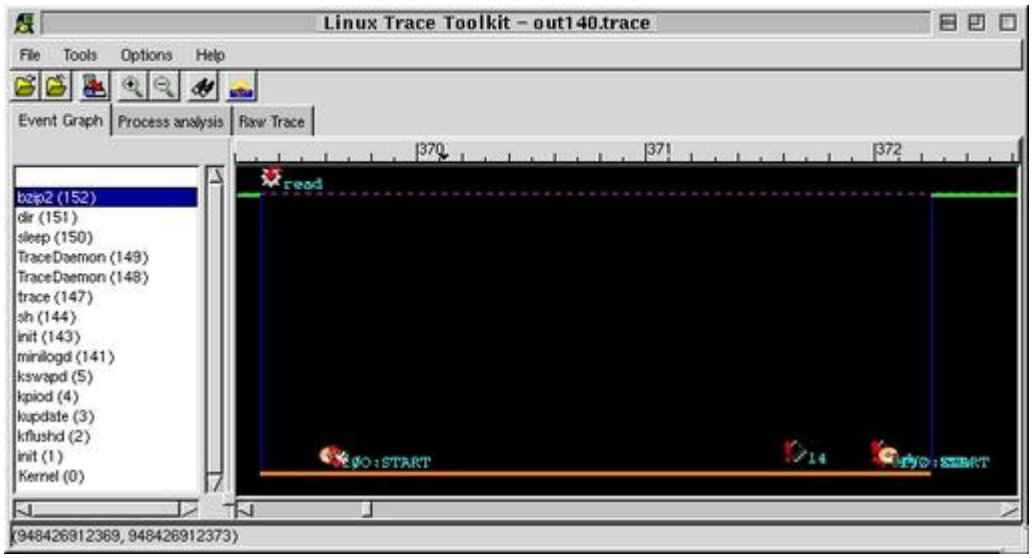


Figure 2. Event Graph

Giving a different perspective, the second thumbnail, Process Analysis, provides us with an in-depth analysis of each process that existed during the trace and a global analysis of the system. Figure 3 presents a sample of this analysis. On the left, the process tree is presented. At the top of the tree lies the idle task (PID 0) which is presented as being *The All Mighty*. Since the idle task never does anything worth tracing, it is used to present data about the whole system: trace start time, trace end time, duration, time spent idle (idle was scheduled as running). Then the number of occurrences of key events is presented. Notice some events given here can't be accounted for by using the information in /proc.

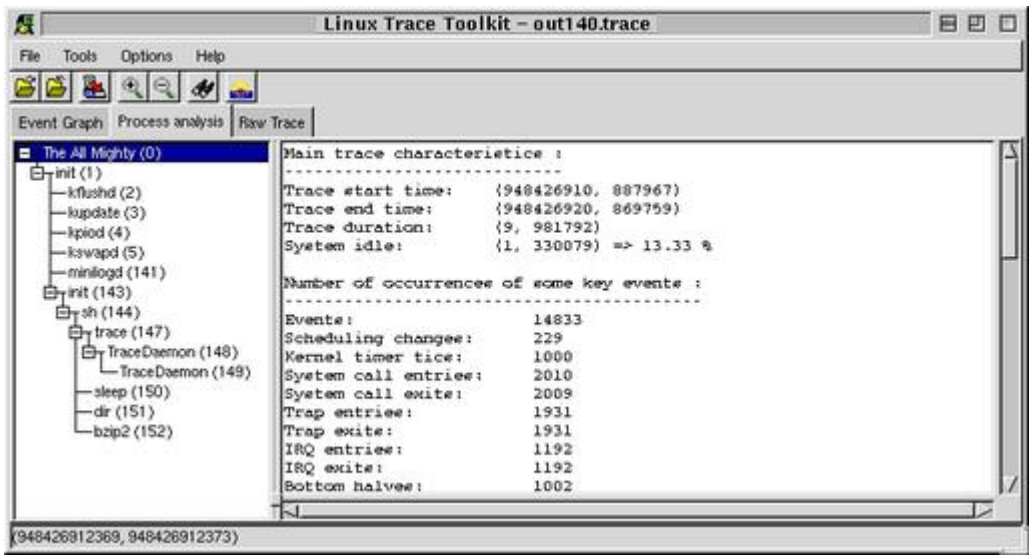


Figure 3. Process Analysis

Listing 1

When selecting dir and bzip2 processes, the information in Listing 1 is displayed. Notice the difference in percentages between the time running (time

scheduled by the kernel as having control over the CPU) and the time spent actually executing code belonging to the application. In bzip2's case, the gap is very small, which accurately describes bzip2's behavior as CPU-intensive; whereas in dir's case, less than half of the time it was scheduled was spent executing its own code, the rest being spent in kernel routines.

Note also the differences between the time spent waiting for I/O and the time running. **dir** spent more time waiting for I/O than it spent executing. **bzip2**, on the other hand, spent little time waiting for I/O compared to the time it spent in running state. This information wasn't collected using sampling on the system clock, as is done currently for the information available through /proc. This information is calculated using the trace information and is therefore the exact information regarding the observed task.

Last but not least, the third thumbnail presents the raw trace. Figure 4 presents a portion of a raw trace. The events are presented in chronological order of occurrence. For each event, the following information is given: the CPU-ID of the processor on which it occurred, the event type, the exact time of occurrence (down to the microsecond), the PID of the task to which it belongs, the size of the entry in the kernel module that was taken in order to record the event in full and the description of the event. Notice how the address from which a system call was made is displayed. This could eventually be used to correlate with data recorded from within the application or simply to know which part of the code made the call. Unlike other tracing methods such as **ptrace**, there is no effect on the application's behavior, except the minimal event-data collection delays.

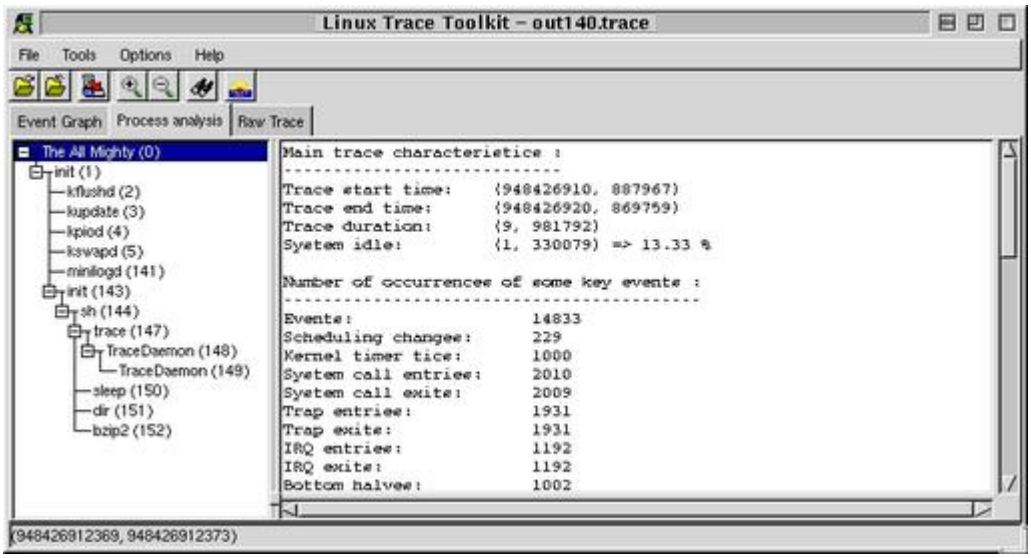


Figure 4. Portion of Raw Trace

LTT's Future

As this article has shown, LTT is an effective tool for recording critical system information. Moreover, it is rather simple to use and the information presented is accessible to a large portion of the community. In an academic environment, LTT can be used in a course on operating systems, helping students get first hand experience with a live operating system and how it interacts with different applications.

Given its capabilities, modularity, extensibility and minimal overhead, we hope to see the tracing code become part of the mainstream Linux kernel soon (maybe not in 2.3/2.4 currently in feature freeze, but in the next development branch). Another possible application for LTT, which gathered a lot of interest, is to use it as part of a security auditing system with Tripwire-type capabilities. At the time of this writing, the authors know of at least one Linux distribution which plans to include LTT as part of their standard distribution.

Karim Yaghmour (karym@opersys.com) is an operating system freak. He's been playing around with OS internals for quite a while and has even written his own OS. He's currently completing his master's degree at the Ecole Polytechnique de Montréal, where the Linux Trace Toolkit is part of his research. He started his own consulting company, Opersys, Inc., that specializes in operating systems (http://www.opersys.com/) and offers expertise and courses on Linux internals and real-time derivatives.

Michel Dagenais (michel.dagenais@polymtl.ca) is a professor at Ecole Polytechnique de Montréal. He has authored or co-authored a large number of scientific publications in the fields of software engineering, structured documents on the Web and object-oriented distributed programming for collaborative applications.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

The Linux Signals Handling Model

Moshe Bar

Issue #73, May 2000

Communication is the key to healthy relationships between threads and the kernel; these are the signals they use to communicate.

Signals are used to notify a process or thread of a particular event. Many computer science researchers compare signals with hardware interrupts, which occur when a hardware subsystem, such as a disk I/O (input/output) interface, generates an interrupt to a processor when the I/O completes. This event in turn causes the processor to enter an interrupt handler, so subsequent processing can be done in the operating system based on the source and cause of the interrupt.

UNIX guru W. Richard Stevens aptly describes signals as software interrupts. When a signal is sent to a process or thread, a signal handler may be entered (depending on the current disposition of the signal), which is similar to the system entering an interrupt handler as the result of receiving an interrupt.

Operating system signals actually have quite a history of design changes in the signal code and various implementations of UNIX. This was due in part to some deficiencies in the early implementation of signals, as well as the parallel development work done on different versions of UNIX, primarily BSD UNIX and AT&T System V. James Cox, Berny Goodheart and W. Richard Stevens cover these details in their respective well-known books, so they don't need to be repeated here.

Implementation of correct and reliable signals has been in place for many years now, where an installed signal handler remains persistent and is not reset by the kernel. The POSIX standards provided a fairly well-defined set of interfaces for using signals in code, and today the Linux implementation of signals is fully POSIX-compliant. Note that reliable signals require the use of the newer **sigaction** interface, as opposed to the traditional **signal** call.

The occurrence of a signal may be synchronous or asynchronous to the process or thread, depending on the source of the signal and the underlying reason or cause. Synchronous signals occur as a direct result of the executing instruction stream, where an unrecoverable error (such as an illegal instruction or illegal address reference) requires an immediate termination of the process. Such signals are directed to the thread which caused the error with its execution stream. As an error of this type causes a trap into a kernel trap handler, synchronous signals are sometimes referred to as traps.

Asynchronous signals are external to (and in some cases, unrelated to) the current execution context. One obvious example would be the sending of a signal to a process from another process or thread via a kill(2), _lwp_kill(2) or sigsend(2) system call, or a thr_kill(3T), pthread_kill(3T) or sigqueue(3R) library invocation. Asynchronous signals are also aptly referred to as interrupts.

Every signal has a unique signal name, an abbreviation that begins with **SIG** (**SIGINT** for interrupt signal, for example) and a corresponding signal number. Additionally, for all possible signals, the system defines a default disposition or action to take when a signal occurs. There are four possible default dispositions:

- Exit: forces the process to exit.
- Core: forces the process to exit and create a core file.
- Stop: stops the process.
- Ignore: ignores the signal; no action taken.

A signal's disposition within a process's context defines what action the system will take on behalf of the process when a signal is delivered. All threads and LWPs (lightweight processes) within a process share the signal disposition, which is processwide and cannot be unique among threads within the same process.

Table 1

Table 1 provides a complete list of signals, along with a description and default action. The data structures in the kernel to support signals in Linux are to be found in the task structure. Here are the most common elements of said structure pertaining to signals:

- **current-->*sig*** are the signal handlers.
- **sigmask_lock** is a per-thread spinlock which protects the signal queue and atomicity of other signal operations.
- **current-signal** and **current-blocked** contain a bitmask (currently 64 bits long, but freely expandable) of pending and permanently blocked signals.

- **sigqueue** and **sigqueue_tail** is a double-linked list of pending signals—Linux has RT signals which can be queued as well. “Traditional” signals are internally mapped to RT signals.

Signal Description and Default Action

The disposition of a signal can be changed from its default, and a process can arrange to catch a signal and invoke a signal-handling routine of its own or ignore a signal that may not have a default disposition of **Ignore**. The only exceptions are **SIGKILL** and **SIGSTOP**; their default dispositions cannot be changed. The interfaces for defining and changing signal disposition are the signal and sigset libraries and the **sigaction** system call. Signals can also be blocked, which means the process has temporarily prevented delivery of a signal. Generation of a signal that has been blocked will result in the signal remaining as pending to the process until it is explicitly unblocked or the disposition is changed to **Ignore**. The **sigprocmask** system call will set or get a process's signal mask, the bit array inspected by the kernel to determine if a signal is blocked or not. **thr_setsigmask** and **pthread_sigmask** are the equivalent interfaces for setting and retrieving the signal mask at the user-threads level.

I mentioned earlier that a signal may originate from several different places for a variety of different reasons. The first three signals listed in Table 1—**SIGHUP**, **SIGINT** and **SIGQUIT**—are generated by a keyboard entry from the controlling terminal (**SIGINT** and **SIGHUP**) or if the control terminal becomes disconnected (**SIGHUP**—use of the **nohup** command makes processes “immune” from hangups by setting the disposition of **SIGHUP** to **Ignore**).

Other terminal I/O-related signals include **SIGSTOP**, **SIGTTIN**, **SIGTTOU** and **SIGTSTP**. For the signals originating from a keyboard command, the actual key sequence that generates the signals, usually **CTRL-C**, is defined within the parameters of the terminal session, typically via **stty(1)** which results in a **SIGINT** being sent to a process, and has a default disposition of **Exit**.

User tasks in Linux, created via explicit calls to either **thr_create** or **pthread_create**, all have their own signal masks. Linux threads call **clone** with **CLONE_SIGHAND**; this shares all signal handlers between threads via sharing the **current->sig** pointer. Delivered signals are unique to a thread.

In some operating systems, such as Solaris 7, signals generated as a result of a trap (**SIGFPE**, **SIGILL**, etc.) are sent to the thread that caused the trap. Asynchronous signals are delivered to the first thread found not blocking the signal. In Linux, it is almost exactly the same. Synchronous signals happening in the context of a given thread are delivered to that thread.

Asynchronous in-kernel signals (e.g., asynchronous network I/O) is delivered to the thread that generated the asynchronous I/O. Explicit user-generated signals get delivered to the right thread as well. However, if **CLONE_PID** is used, all places that use the PID to deliver a signal will behave in a “weird” way; the signal gets randomly delivered to the first thread in the **pidhash**. Linux threads don't use **CLONE_PID**, so there is no such problem if you are using the pthreads.h thread API.

When a signal is sent to a user task, for example, when a user-space program accesses an illegal page, the following happens:

- **page_fault** (entry.S) in the low-level page-fault handler.
- **do_page_fault** (fault.c) fetches i386-specific parameters of the fault and does basic validation of the memory range involved.
- **handle_mm_fault** (memory.c) is generic MM (memory management) code (i386-independent), which gets called only if the memory range (VMA) exists. The MM reads the page table entry and uses the VMA to find out whether the memory access is legal or not.

Listing 1

The case we are interested in now is when the access was illegal (e.g., a write was attempted to a read-only mapping): handle_mm_fault returns 0 to do_page_fault in this case. As you can see from Listing 1, locking of the MM is very finely grained (and it better be this way); the **mm->mmap_sem**, per-MM semaphore, is used (which typically varies from process to process).

force_sig(**SIGBUS**,**current**) is used to “force” the **SIGBUS** signal on the faulting task. force_sig delivers the signal even if the process has attempted to ignore **SIGBUS**.

force_sig fills out the signal event structure and queues it into the process's signal queue (**current->sigqueue** and **current->sigqueue_tail**). The signal queue holds an indefinite number of queued signals. The semantics of “classic” signals are that follow-up signals are ignored—this is emulated in the signal code kernel/signal.c. “Generic” (or RT) signals can be queued arbitrarily; there are reasonable limits to the length of the signal queue.

The signal is queued, and **current-signal** is updated. Now comes the tricky part: the kernel returns to user space. Return to user space happens from **do_page_fault=>page_fault (entry.S**), then the low-level exit code in entry.S is executed in this order:

```
page_fault=>(called do_page_fault)=>error_code=>
ret_from_exception=>(checks if return to user space)=>
```

```
ret_with_reschedule=>(sees that current->signal is nonzero)
=>calls do_signal
```

Next, **do_signal** unqueues the signal to be executed. In this case, it's **SIGBUS**.

Then **handle_signal** is called with the “unqueued” signal (which can potentially hold extra event information in case of real-time signals/messages).

Next called is **setup_frame**, where all user-space registers are saved and the kernel stack frame return address is modified to point to the handler of the installed signal handler. A small sequence of code jumper is put on the user stack (obviously, the code first makes sure the user stack is valid) which will return us to kernel space once the signal handler has finished. (See Listing 2.)

Listing 2

Careful: this area is one of the least-understood pieces of the Linux kernel, and for good reason; it is really tough code to read and follow.

The **popl %eax ; movl $,%eax ; int $0x80** x86 assembly sequence calls **sys_sigret**, which later on will restore the kernel stack frame return address to point to the original (faulting) user address.

What is all this magic good for? Well, first the kernel has to guarantee that signal handlers get called properly and the original state is restored. The kernel also has to deal with binary compatibility issues. Linux guarantees that on the IA-32 (Intel x86) architecture, we can run any iBC86-compliant binary code. Speed is also an issue.

Listing 3

Finally, we return to entry.S *again*, but **current-signal** is already cleared, so we do not execute do_signal but jump to **restore_all** as shown in Listing 3. **restore.all** executes the “iret” that brings us into user space. Suddenly, we are magically executing the signal handler.

Did you get lost yet? No? Here is some more magic. Once the signal handler finishes (it does an assembly “ret” like all well-behaving functions), it will execute the small jumper function we have set up on the user stack. Again we return to the kernel, but now we execute the **sys_sigreturn** system call, which lives in arch/i386/kernel/signal.c as well. It essentially executes the following code section:

```
if (restore_sigcontext(regs, &frame->sc, &eax))
     goto badframe;
return eax;
```

The above code restores the exact user-register contents into the kernel stack frame (including the return address and flags register) and executes a normal **ret_from_syscall**, bringing us back to the original faulting code. Hopefully the **SIGBUS** handler has fixed the problem of why we were faulting.

Now, while reading the above description, you might think this is awfully complex and slow. It actually isn't; **lmbench** reveals that Linux has the fastest signal-handler installation and execution performance by far of any UNIX running:

```
moon:~/l> ./lat_sig install
Signal handler installation: 1.688 microseconds
moon:~/l> ./lat_sig catch
Signal handler overhead: 3.186 microseconds
```

Best of all, it scales linearly on SMP:

```
moon:~/l> ./lat_sig catch & ./lat_sig catch &
Signal handler overhead: 3.264 microseconds
Signal handler overhead: 3.248 microseconds
moon:~/l> ./lat_sig install & ./lat_sig install &
Signal handler installation: 1.721 microseconds
Signal handler installation: 1.689 microseconds
```

Signals and Interrupts, A Perfect Couple

Signals can be sent from system calls, interrupts and bottom-half handlers (see sidebar) alike; there is no difference. In other words, the Linux signal queue is interrupt-safe, as strange and recursive as that sounds, so it's fairly flexible.

Bottom-Half Handlers

An interesting signal-delivery case, however, is on SMP. Imagine a thread is executing on one processor, and it gets an asynchronous event (e.g., synchronous socket I/O signal) from an IRQ handler (or another process) on another CPU. In that case, we send a cross-CPU message to the running process, so there is no latency in signal delivery. (The speed of cross-CPU delivery is about five microseconds on a Pentium II 350MHz.)

Conclusions

Once again, we notice how Linux is actually the technology leader in important kernel aspects such as scheduling, interrupt handling and signals handling. This also proves the conjecture that the Linux developer community is collectively more capable and more resourceful than any private corporation's R&D department could ever be.

Resources

Moshe Bar (moshe@moelabs.com) is an Israeli system administrator and OS researcher, who started learning UNIX on a PDP-11 with AT&T UNIX Release 6 back in 1981. He holds an M.Sc. in computer science. Visit Moshe's web site at http://www.moelabs.com/.

Archive Index Issue Table of Contents

Advanced search

Linux at Yorktown High School

Justin Maurer

Issue #73, May 2000

How this school is utilizing Linux to teach students, do remote administration and save money.

Yorktown High School in Arlington, Virginia is composed of about 1500 students in grades 9-12. More than 80 of these students are enrolled in a computer science course, and all of them use Linux.

Until the 1997-98 school year began, these courses were taught using proprietary software from Microsoft, Borland and other vendors. Since then, Linux has grown steadily within the computer science department and throughout the rest of the school.

Computers are in every classroom at the school. The variety of machines ranges from Apple Macintoshes used by the newspaper, yearbook and English classes, to x86-class machines, a Sun workstation, and a PowerMac clone used by the computer science classes. Yorktown is connected to the Internet via a T1 line to the county Education Center. Internally, the school is networked via 10baseT Ethernet, with multiple drops (Ethernet ports) available in every room.

Three separate computer science classes are offered: Computer Science, Advanced Placement Computer Science and Advanced Topics.

Computer Science (http://yhslug.tux.org/csc/)

Computer Science is taught by Jeff Elkner. This class uses Pentium-class machines, all running Linux Mandrake 6.1. Students are greeted with a graphical **kdm** login screen which allows them to click on their photo (taken with a digital camera at the beginning of the year) and enter their password to log in. Students may use any machine in the lab, because we have configured all the systems to use NIS and NFS. The machines are served by our shell server, named linus.

Until this year, students learned to program in C++ with the GNU C++ compiler (g++). This year, students begin with Python and move to C++ only after mastering the fundamentals of programming. This was partly inspired by the fact that Guido van Rossum, Python's creator, has agreed to visit the school and help these students as part of his Computer Programming for Everyone (CP4E) project.

Advanced Placement Computer Science

(http://yhslug.tux.org/csc/adv/)

AP Computer Science is also taught by Jeff Elkner. Students continue to learn C++, as well as take AP C++ classes provided by the College Board. This has been problem-free, with one small exception. Though the College Board's AP classes are platform-independent, they are not free software. Former student Paul Morie (morie@uiuc.edu) and current student Jonah Cohen (ComAsYuAre@aol.com) collaborated to write a free replacement called **pclasses**. This software is available under the GPL and can be downloaded from the school's web site.

Advanced Topics

Advanced Topics students have no set course of study. At the beginning of each quarter, they outline a set of goals and must meet them before the beginning of the next term. Students usually work in small groups to accomplish their projects.

One project that has been very successful is the Python Resource Kit (http://yhslug.tux.org/python/). The kit is a downloadable ISO image, as it is meant for redistribution, but the school often burns copies for first-year students so that they can take them home and install Python on their Windows machines.

Another project that has made much progress so far is the Open Book Project (http://yhslug.tux.org/obp/). The project is an effort to create a freely redistributable textbook for computer science teachers. It is based on Allen Downey's *How to Think Like a Computer Scientist* book, and has been modified for Python and C++ instead of Java.

Successes

Because the county had difficulties in setting up an e-mail server to offer accounts to students and teachers, we have configured a mail server and given free e-mail accounts to students and teachers who have requested them. In addition to mail, Linux also powers the school's web server (http://yhspatriot.yorktown.arlington.k12.va.us/).

Another use of Linux at Yorktown is powering the Student Portfolio (http://yhslug.tux.org/~portfolio/cgi-bin/) database. Over the summer, Lex Bereznhy collaborated with me to write a Python-based database that allows students to plan their schedules, showcase their work and communicate with their teachers.

Another one of the school's successful projects is PyTicket (http://yhslug.tux.org/~pyticket/cgi-bin/). Teachers or anyone else in need of assistance can simply fill out a form and get a web-based "ticket". Students check these tickets every morning, and are dispatched to help teachers with their computer problems.

Using Linux has saved the county tens of thousands of dollars in software licenses, provided students with an opportunity to learn Linux/UNIX basics, and has given the school an opportunity to contribute to the free software community.

Finally, this article would not be complete without mention of the school's newest project. By the time you read this, the inaugural meeting of the Yorktown High School Linux Users Group (YHSLUG, http://yhslug.tux.org/) will have been held. Attendance of around 20 users is expected. Drop by if you are in the neighborhood!

email: justin@slashdot.org

Justin Maurer is the youngest hacker at Helix Code, Inc. He formerly wrote for Slashdot, and now sometimes contributes to Linux.com. He is also a Debian developer, and can be reached via e-mail at justin@slashdot.org.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Rapid Program-Delivery Morsels, RPM

Marcel Gagné

Issue #73, May 2000

Installing and upgrading software need not be difficult—Monsieur Gagné tempts us with delectable RPM.

François! Why did you not tell me immédiatement that these distinguished diners had arrived? Run along, now. Vite! Bring the '82 Margaux from the cellar.

Forgive me, mes amis, for not welcoming you sooner. François does not like to disturb me when I sample, or rather, take inventory of the wine cellar. Please, sit down, and welcome once more to Chez Marcel. As you know, this month's issue features delightful forays into the world of software development.

Installing and distributing software can be a difficult task. For the, shall we say, not-so-technical users, untarring and unzipping source, compiling, installing and keeping track of all that has been done can be daunting. Upgrading that software once the deed is done can be another trial. This is what a package manager seeks to simplify. The most popular and versatile of these is RPM, the Red Hat Package Manager. With RPM, you can make managing your distribution an easy process for the end user by transforming the whole process of untarring, compiling, installing and cleaning up into a single, one-line command. RPM even maintains a database of installed products so that you can verify installed software for completeness or query file locations.

A number of Linux distributions come with RPM as their package manager of choice. You'll find RPM used by Red Hat, Caldera, Mandrake and TurboLinux, just to name a few. Even if your Linux distribution does not come with RPM, you can still install and use it. What's even more exciting is that RPM will run on a number of other UNIX flavors as well. These include AIX, HP-UX, IRIX and others. I can hear the ads now: “The Red Hat Package Manager . . . it's not just for Red Hat anymore.”

Your taste buds are proving Monsieur Pavlov correct, non? You want to try your own hand at distributing your software using RPM, but you do not know where to begin? Follow me into the kitchen and I will show you a simple recipe for creating your own RPMs.

Let's pretend I want to distribute a clever little package called **gatekeeper**, version 1.0. This neat little piece of *hypothetical* software comes with a web interface for authentication, a CGI script which calls a small setuid program, which then calls another Perl script to update /etc/hosts.allow in order to grant access. The installed files are as follows:

```
/home/httpd/html/gatekeeper/keypad.html
/home/httpd/cgi-bin/gatekeeper/locksmith.pl
/usr/local/.Admin/sethosts.pl
/usr/local/bin/suguard
```

My setuid program is **suguard** and will therefore install with setuid permissions, while the .pl files will require standard executable permissions for CGI scripts. We should perhaps add a README to go with this collection. We also want to release this software under the GNU Public License. So, where do we go from here?

The first step in creating your new RPM is to build a .spec file, the basic configuration file for RPM. It is essentially a recipe that RPM uses to build your final package. As with any recipe, you have some flexibility in terms of what can be added or removed, but there are basics that should be followed. After all, if you make *Tourtière* with apples instead of meat, you have apple pie with vegetables and spices and it just does not work. Even François may not be able to provide you with enough wine to wash that down.

We will now cover the details of this recipe, the spec .file. In my case, it is called gatekeeper-1.0-*x*.spec and is broken up into sections addressing specific characteristics of the distribution package to be built. The important sections are the “preamble”, “%prep”, “%build”, “%install” and “%files”. The preamble is a description of the package: release information, who built it and where to obtain the source. As the name implies, this is located right at the top of my .spec file. Unlike the other sections, it has no header—you simply begin the file with it. For my gatekeeper program, I have the following preamble:

```
Summary: Allows authorized clients firewall access
Name: gatekeeper
Version: 1.0
Release: 1
Copyright: GPL
Group: Applications/Networking
Source: ftp://ftp.mycompany.com/pub/packages/gatekeeper-1.0.tar.gz
URL: http://www.mycompany.com/
Packager: Chef Marcel
ExclusiveArch: i386
```

We begin with a simple summary description of the package. This is followed by the name, a version number, and a release number. If you have ever downloaded or installed with RPM, you are familiar with what is happening here. My package, when complete, will be named gatekeeper-1.0-1.i386.rpm. I include a copyright statement, the source location (where the package can be downloaded using FTP), a URL for documentation on the package, and the chef who put the whole package together is mentioned as “Packager”.

For an example of where this becomes useful, try giving the command below to any package on your system. For example, let's query the “net-tools” package:

```
rpm -qi net-tools
```

Here is the output from that command.

```
Name: net-tools
Relocations: (not relocateable)
Version: 1.53
Vendor: Red Hat Software
Release: 1
Build Date: Sun 29 Aug 1999
08:16:43 PM EDT
Install date: Mon 17 Jan 2000 05:07:51 PM EST
Build Host: porky.devel.redhat.com
Group: System Environment/Base
Source RPM: net-tools-1.53-1.src.rpm
Size: 569756
License: GPL
Packager: Red Hat Software http://developer.redhat.com/bugzilla
Summary: Basic tools for setting up networking:
Description: The net-tools package contains the
basic tools needed for setting up networking:
arp, rarp, ifconfig, netstat, ethers and route.
```

Next, in my “.spec” file, I include a **%description** heading. In this section, I include a somewhat more descriptive explanation of the package than my simple one-liner at the top of the last section. In a way, this is part of the first section, in that it makes up the total package description.

```
%description
Allows authorized clients firewall access
What makes this so cool is that clients with
dynamic IP addresses can still be allowed access
to network resources with proper authentication.
After a set inactivity time (fifteen minutes),
access is deleted from the system.
```

So far, we have a lot of information, but now the real work begins. If you look at this next section closely (**%prep**), you'll see it is just a place to put commands that prepare your sources (or in my examples, a few scripts) for building. You could conceivably just call a shell script to do everything, or list the commands specifically. You'll see it is fairly simple.

```
%prep
rm -rf $RPM_BUILD_DIR/gatekeeper-1.0
zcat $RPM_SOURCE_DIR/gatekeeper-1.0.tar.gz |
tar -xvf -
```

Some of these section headers are actually powerful macros. For instance, the **%setup** header for the next section can do a fair bit without being given any arguments. (In exactly the same way I did not! Very clever, non?) What this macro does without any help is unpack the sources (the tar.gz file) and move (using cd) into that directory—very nice. You can also add some flags to this macro to customize your build. Today, we'll keep things simple:

```
%setup
```

While we look at this lonely but powerful **%setup** macro, I should point out that there is another. A **%patch** macro is available to help you easily apply patches to your build. Next in line is the **%build** section:

```
%build
```

Ah, yes, another lonely section. François, could you pour a glass of wine for our build section to help it through its loneliness? Sit down, François; I was only joking. What we have here is a section that is open to any shell commands you may need to use in order to build your package. For instance, if I was working with a source package, I could have something like this:

```
%build
ftl_drive_preparations
make depend
make
```

Obviously, this would be a much higher tech package than I am building here, non?

As we move down the .spec file, the demands for detail increase. For instance, the **%install** section contains a list of commands necessary to install the package: any directory creation commands, install commands and anything else required. You could also simply call another shell script or a **make install**. I prefer my .spec file to reflect what happens with my build. Note the backslashes for lines that were too long to fit in this example.

```
%install
mkdir -p $RPM_BUILD_ROOT/home/httpd/html/gatekeeper
mkdir -p $RPM_BUILD_ROOT/home/httpd/cgi-bin/gatekeeper
mkdir -p $RPM_BUILD_ROOT/usr/local/.Admin
mkdir -p $RPM_BUILD_ROOT/usr/local/bin
install -m 644 home/httpd/html/scidns/keypad.html\
$RPM_BUILD_ROOT/home/httpd/html/scidns/keypad.html
install -m 755 home/httpd/cgi-bin/scidns/locksmith.pl
$RPM_BUILD_ROOT/home/httpd/cgi-bin/scidns/locksmith.pl
install -m 755 usr/local/.Admin/sethosts.pl \
$RPM_BUILD_ROOT/usr/local/.Admin/sethosts.pl
install -m 4755 usr/local/bin/suguard \
$RPM_BUILD_ROOT/usr/local/bin/suguard
```

Almost there, mes amis. When we have finished putting our RPM binary together, we should clean up after ourselves, non? This is where we do it. This is the **%clean** section.

```
%clean
rm -rf $RPM_BUILD_ROOT
```

Now we come to our final section, **%files**. This is simply a list of all files that make up your package. Keep in mind that the files must be listed here—anything missing in this section will not make it into the final RPM.

```
%files
%doc README
/home/httpd/html/gatekeeper/keypad.html
/home/httpd/cgi-bin/gatekeeper/locksmith.pl
/usr/local/.Admin/sethosts.pl
/usr/local/bin/suguard
```

While you are analyzing the wonderful simplicity of this section, note also the **%doc** parameter. The file I include here will automatically get installed under the /usr/doc hierarchy on my system to represent the package. In this case, the directory would be /usr/doc/gatekeeper-1.0 with one file (README) sitting underneath it.

Now comes the moment we have all been waiting for. Your .spec file is complete, and you long to see this work. This is how you build your package:

```
rpm -bb gatekeeper-1.0-.spec
```

The **-bb** option means "build binary". The rpm command has many flags that define whether you are building source or binaries, or executing an individual section of the .spec file. When you press **ENTER** on the above command, many things start scrolling past your eyes as RPM deals with every section of your .spec file. The line I look for in this case is

```
Wrote: /usr/src/redhat/RPMS/i386/gatekeeper-1.0-1.i386.rpm
```

That, mes amis, is a beautiful thing.

Throughout this article, I have referred to variables like **RPM_BULD_ROOT** without explaining where these things come from. In the case of a Red Hat system like mine, this is /usr/src/redhat where I find the following directories: BUILD, RPMS, SOURCES, SPECS and SRPMS. If you are curious as to how they are defined, have a look through the file /usr/lib/rpm/macros. If you cannot find the "macros" file in this location (remember, rpm must be loaded), use this command to locate it:

```
rpm -ql rpm | grep macros
```

You see, rpm is a wonderful tool for so many things, including working with rpm itself.

What? Ah, yes. I hear you, mes amis. You are asking, "Marcel, is there no other way to build an RPM, something perhaps a little friendlier?" Do not despair.

Chef Marcel has a particular fondness for that well-aged UNIX editor, **vi**. Nevertheless, there are other ways to build an RPM package that bring the process to the desktop. Allow me to show you these delicacies.

The first item I will mention is something from KLabs called (strangely enough) **RPM Builder** (see Figure 1). This is a Tcl/Tk application (written by *Seek3r*) that automatically builds the framework for a .spec file from a tar.gz distribution. Very neat, despite the fact that our friends at KLabs built the screen based on a 1024x768 screen. On my 800x600 screen, I lose part of it at the bottom of my notebook. There is no need to tell them—they already know about it. Still, it is worth checking out.

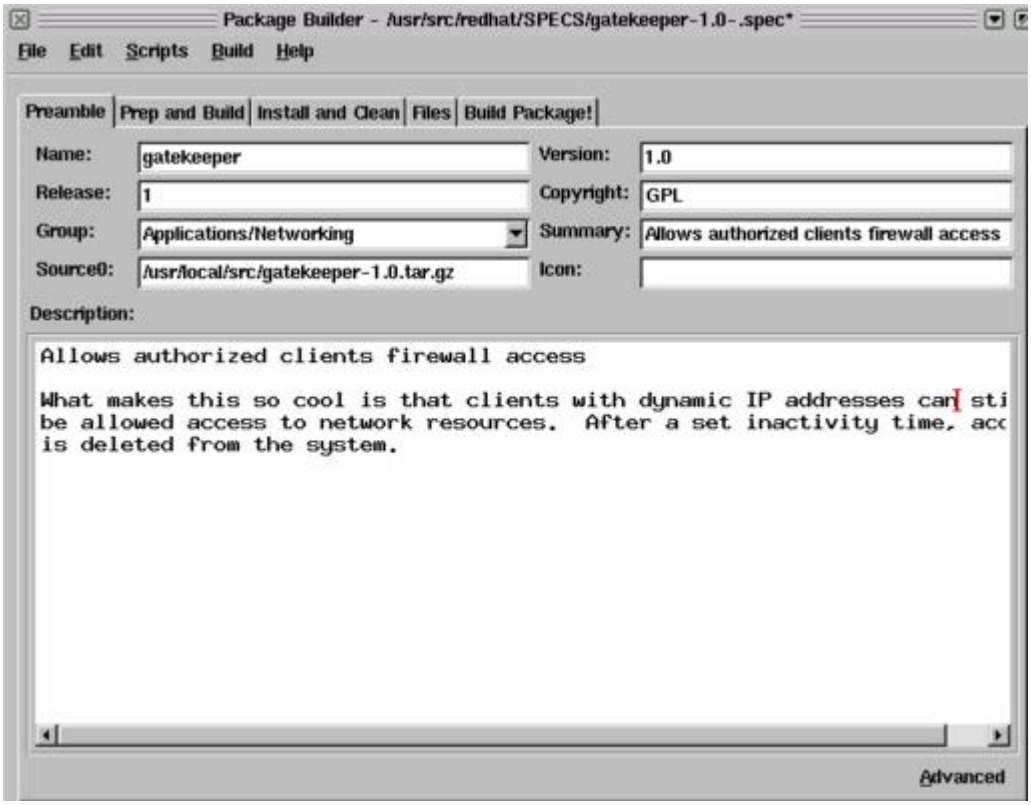


Figure 1. RPM Builder

Another package I would recommend is **pkgbuild** (see Figure 2) from Davin S. Hills of the Hills Design Group. This is a very nice-looking package with a screen design that breaks the GUI into sections not unlike the RPM-build process itself. You have a tab for the preamble, one for “Prep and Build”, “Install and Clean” and so on. Everything can be done from one interface. The software is built on the FOX C++ development libraries written by Jeroen van der Zijp, so you will need to pick that up if you want to build the package yourself. RPMs of pkgbuild are available for Red Hat users.

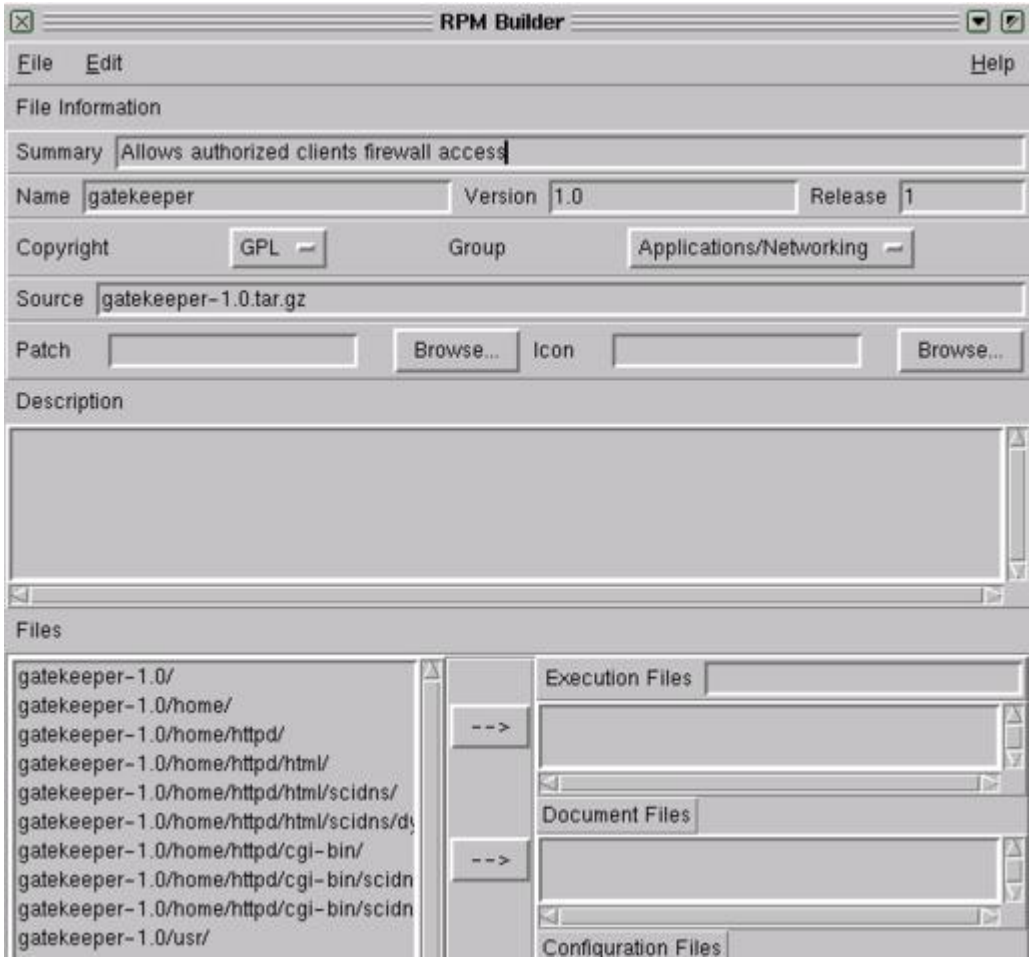


Figure 2. pkgbuild

Make sure you check out the RPM resources in the sidebar. For more RPM masterpieces, I invite you to visit the official RPM web site for all things RPM, and the RPM Documentation Project. The RPM Documentation Project web site is in its infancy and somewhat sparse. The maintainers promise much more information in the near future, so by the time you read this, it may be *brimming* with information. Check it out anyhow. You will find a link to the *Maximum RPM* book by Edward C. Bailey, free and on-line. This edition of the book is a couple of years old, but you will still find lots of useful information.

Well, mes amis, that dreadful rooster outside is telling me it is closing time. I sincerely hope I have managed to assist you in bringing your brilliant Linux software package a little closer to reality and a little closer to simple, reproducible installations and updates. Au revoir, mes amis. Remember, you are always welcome here at *Chez Marcel*. Bon Appétit!

Resources

email: mggagne@salmar.com

Marcel Gagné (mggagne@salmar.com) lives in Mississauga, Ontario. In real life, he is president of Salmar Consulting Inc., a systems integration and network consulting firm. He is also a pilot, writes science fiction and fantasy and edits TransVersions, a science fiction, fantasy and horror magazine. He loves Linux and all flavors of UNIX and will even admit it in public. Check out his System Administration column on *LJ*'s web site.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Creating Queries

Reuven M. Lerner

Issue #73, May 2000

Don't be afraid of large joins—learn to generate complex SQL queries from easy-to-use interfaces.

Over the last few years increasing numbers of web sites have begun to integrate a relational database management system (RDBMS). Databases excel at storing and retrieving information quickly and easily and make it possible for web developers to create sophisticated applications without getting bogged down in the details.

While it is not hard to integrate a database into a web site, it can sometimes be tricky to design the database tables, as well as construct sophisticated applications to use them.

Last month we looked at (normalization) the technique that makes it possible to reduce potential errors while increasing the speed and flexibility of our database. This makes it easier to create a variety of applications with the same data. This comes at a price: the queries needed to retrieve information from the database become more complex, often joining data from two, three or four tables. With a bit of training, however, programmers can easily overcome their fear of large joins, using them to retrieve only the information they need.

Most users are not programmers, and it is unrealistic to expect them to construct complex queries on their own. The trick is to create a user interface that makes it possible to answer a large subset of the possible questions, without overwhelming users with a variety of options.

This month, we will spend some time looking at how to generate complex SQL queries from relatively easy-to-use interfaces. Our examples will draw on the example tables from last month, which describe the Israeli train system.

In the end, we will write two types of programs. Some programs (often known as “query generators” in the database trade) will generate HTML forms that can be used to create queries. Other programs will translate the HTML form into an actual SQL query, displaying the results in the user's browser.

Limiting Options

Linux users often sneer at the Macintosh interface, because it limits the available options. At the bash prompt, a Linux user can invoke many thousands of commands. Furthermore, the output of any program can be piped to a file or another program. This multitude of options makes Linux a particularly powerful system, yet one that is difficult to learn and master.

Most users are not interested in power, but simply want to get their work done. Giving users too much flexibility can sometimes be a hindrance. In such cases, we want to limit users' options, forcing them to make a choice that our programs can handle. This is one of the key advantages of a GUI, it reduces the chance for user error by reducing the number of available options. It also reduces the number of potential inputs a program might receive, making it easier to test applications.

For example, consider an HTML form that asks users to enter their destination train station. Many HTML forms use a text area to input such information as follows:

```
<input type="text" name="destination" size="30">
```

Allowing users to enter a station name in this way opens Pandora's box, forcing the program to handle misspellings, capitalization issues and even problems stemming from whitespace. Databases might be good at many things, but they do require precise inputs that match exactly.

We can improve the situation and simplify our program by limiting the user's actions with a <select> list. Each <option> in this <select> list will correspond to one row from the RailStations table, with the **value** attribute set to the row's “ID” and the user-visible text set to the **name** attribute. For example, we could do the following:

```
<select name="destination">
<option value="1">Nahariya
<option value="2">Akko
<option value="3">Hof Hacarmel
<option value="4">Tel Aviv Central
<option value="5">Tel Aviv Hashalom
<option value="6">Lod
<option value="7">Rehovot
<option value="8">Herzliya
</select>
```

The above <select> list provides more reliable input than a text field. However, it presents several problems. First and foremost, placing the above static HTML in a file means the RailStations table and the <select> list will inevitably go out of synch. From the user's perspective, the above <select> list is difficult to use, because it orders the items according to their ID numbers rather than their station names.

We can solve both of these problems with our query generator, a CGI program that produces HTML forms based on information in the database. Listing 1 (see Resources) contains a simple non-CGI program, **select-list-from-table.pl**, that produces the above <select> list based on the current contents of the RailStations table with the stations listed in alphabetical order. If a train station's name is changed or a new station is added, an HTML form created by select-list-from-table.pl will immediately reflect the new value.

Because rows returned by a **SELECT** statement can be returned in any order specified by the **ORDER BY** clause, it is possible to produce a <select> list in an order other than alphabetical. For example, users might prefer to see a list of train stations by location.

Leaving Options Open

The <select> list produced by Listing 1 has at least one problem. What if the user doesn't care about the particular piece of information in the <select> list? For example, assume the user is interested in traveling from Tel Aviv to Rehovot at some point during the day, but has not yet decided when. Forcing a user to choose a time from a <select> list makes things more difficult than they should be. It would be nice to have an “any” option on each <select> list, allowing users to indicate this particular field can have any information.

Implementing this strategy requires two things. First, one <option> value can be guaranteed not to correspond to an existing row's primary key. Second, the CGI program creating the final query will identify this value and modify the SQL accordingly.

Luckily, the combination of MySQL and Perl works quite well in this context. MySQL's auto-incrementing primary keys begin with 1 and increase until the maximum value is reached. Because an auto-incrementing primary key can never be zero, we can create an additional <option> line in the <select> list:

```
<option value="0" selected>Any
```

By making this value 0, we ensure it cannot correspond to any actual row's primary key. And by marking it “selected”, we set this to be the default value for the <select> menu. New visitors to the site who accept the default values will

thus get the widest possible search, with the fewest possible **WHERE** clauses. Each selection of an <option> with a non-zero value will add a new **WHERE** clause to the resulting SQL.

Constructing Basic Queries

The following query lists all times at which trains from Nahariya (ID 1) have an endpoint of Tel Aviv (ID 5):

```
SELECT S.name, DT.departure_time
FROM RailStations S, DepartureTimes DT, Trains T,
     StationLines SL
   WHERE T.id = DT.train_id
      AND T.line_id = SL.line_id
      AND S.id = 1
      AND SL.station_id = DT.station_id
      AND DT.station_id = S.id
      AND T.destination_id = 5
   ORDER BY DT.departure_time
;
```

We can turn the above query into a CGI program by replacing the two ID numbers with placeholder values. If we fill in the placeholders with the contents of the “origin” and “endpoint” HTML form elements, we can find the times of all trains from one station heading toward a particular endpoint. Such a CGI program, **list-trains-to-endpoint.pl**, is in Listing 2 (see Resources).

The above code works just fine until someone selects one of the **Any** items with a value of 0. If that happens, MySQL will not return any rows from the **SELECT**, because no stations have an ID of 0. The solution is to make those parts of the query conditional, inserting them only if an actual value was indicated.

We accomplish this by placing two “if” statements in the middle of the code that assembles the SQL statement. Because the generic ID is 0, our program can test for a set value simply by putting the variable name inside parentheses, which implies a test for non-zero values:

```
my $sql = "SELECT S.name, DT.departure_time ";
$sql .= "FROM RailStations S, DepartureTimes DT, Trains T, StationLines SL ";
$sql .= "  WHERE T.id = DT.train_id ";
$sql .= "   AND T.line_id = SL.line_id ";
if ($origin)
{
$sql .= "   AND S.id = ? ";
push @placeholders, $origin;
}
$sql .= "   AND SL.station_id = DT.station_id ";
$sql .= "   AND DT.station_id = S.id ";
if ($endpoint)
{
$sql .= "   AND T.destination_id = ? ";
push @placeholders, $endpoint;
}
$sql .= " ORDER BY DT.departure_time ";
```

Listing 3 (see Resources) contains a program, **better-list-trains-to-endpoint.pl**, that allows users to specify a station of origin, an endpoint for the train, neither

or both. If the user specifies only the station of origin, the program will display a list of trains departing from that station without regard for direction. If the user specifies only the endpoint, it will list all departures toward that station. Most applications do not need to give such headway to users, and might want to trap inputs in which both elements are assigned values of 0. At the same time, no harm comes from allowing users to amuse themselves with nonsensical queries.

When Does the Train Leave?

The above queries work just fine, yet they ignore a crucial issue when working with train schedules: people typically want to specify the time they prefer to leave or arrive. It would certainly be possible to give users a set of <select> lists corresponding to various hours and minutes in the day, giving them fine control over the assembled query. We could also allow them to enter dates and times in text fields, but as with station names, there are too many possible ways for such input to go wrong.

It is probably easier for them to relate to time information such as “morning”, “afternoon” and “night”, rather than specifying hours. We can do this by using another set of <select> lists, this time specifying hours as values and by setting the hours to the end of the specified period. For example:

```
<select name="time">
<option value="12:00">Morning
<option value="17:00">Afternoon
<option value="21:00">Evening
<option value="24:00">Night
</select>
```

We can find all morning trains by asking for those earlier than 12:00, afternoon trains by asking for those before 17:00 and so forth.

With such a <select> list in place in the query generator form, we can rest assured that users will be able to find their train. If we are concerned that too many trains will match their query, we can add another <select> list to the HTML form, allowing users to limit the number of returned trains. MySQL supports a **LIMIT** clause on **SELECT** queries, making it possible to return a maximum number of rows.

Note that **LIMIT** will always contain a numeric value and can thus be inserted directly into the SQL query without the use of placeholders. Indeed, trying to use placeholders in a **LIMIT** clause will force the number to be quoted, which will cause a MySQL error.

best-list-trains.pl, a program that implements both of these ideas—the “time” element and a “limit” element—is in Listing 4 (see Resources).

Conclusion

While it is tempting to give users infinite freedom to enter information into a web site, it is usually best to limit their inputs as much as possible. Creating simple HTML-based query generators is not difficult and can even be easy once you get the hang of it. The trick is to formulate queries in such a way that the user can get the maximum information while knowing as little as possible about the underlying database. Even when the queries are easy to create, finding ways to turn those queries into language suitable for non-programmers can be a challenge.

Resources

Reuven M. Lerner , an Internet and Web consultant, moved to Modi'in, Israel following his November marriage to Shira Friedman-Lerner. His book Core Perl will be published by Prentice-Hall in the spring. Reuven can be reached at reuven@lerner.co.il. The ATF home page, including archives, source code and discussion forums, is at http://www.lerner.co.il/atf/.

Archive Index Issue Table of Contents

Advanced search

The New and the Old

Jason Kroll

Issue #73, May 2000

Jason takes a look at new commercial offerings for Linux and freeware ported from antique software.

Since this column was first launched, Linux has increased its hoard of commercial games manyfold (not that we cover commercial games here), and the free offerings are sprouting and growing all over the hills and under the hills, some to rather large and sophisticated packages, and all of them expanding the code base across the board. Serious folk often write off games as fluff; after all, computers are serious machines for serious work—work, work, never shirk. Let's be honest, though; games drive the hardware industry, particularly in graphics cards, sound cards and processors. I feel games will help encourage vendors to release their hardware specifications and drivers for Linux—a necessity for a gaming environment, especially more multimedia support. Serious people can moan all they want about how Linux is a server OS for serious applications and how gamers should buzz off, but we know world domination comes through games and multimedia.

Speaking of games and multimedia, for the last couple of months we've been looking at emulating old Arcade and C64 games on Linux, and during that time, there's been a bit of development going on. On the commercial front, we've come a long way since the days of Doom, Quake, and the first Loki mega-port, Civilization: Call to Power. Nowadays, Quake III is the order of the day, and rest assured, it's available for Linux (and runs fine on the Crusoe chips). Loki is about to unleash several new ports, such as Alpha Centauri (the sequel to Civilization, as one might infer), Heroes of Might and Magic III, Heretic II, SimCity 3000 and Soldier of Fortune. The Linux gaming scene can also welcome two relatively new entrants to its niche: MP Entertainment, who delivered the comic-book-quality animated adventure Hopkins FBI (which I *really* intend to review one of these days) and BlackHoleSun Software, which first appeared with the simple game Krilo and is following up with a new project called C4 that looks to be on DVD, if we ever resolve the whole DVD-on-Linux crisis.

Now, even though many of us aren't in love with proprietary software, these games have a benefit in that they draw teenage dollars and interest: dollars to give developers an incentive to support Linux with hardware drivers, and interest to get future hackers into Linux, where we can all benefit from their brain power in years to come. Teenage brains have flexible neurons, probably a necessity for learning a complicated OS. With hardware support and fleets of new hackers, Linux will be loved and supported for years to come. We'll have the whole world running free software. Even though those kids spent their money on video games, we can be playing free offerings like those described here.

We all love antique anything, although if we were part of the neo-conformist, 1950-was-a-good-year martini crowd, we might choose the word “vintage”. Whether it's the ravers going on about the 808/909 vintage sounds, the antiquity Goths scouring antique (er, junk) shops for Victorian trinkets, the nouveau-riche wine tasters, the I-want-to-look-like-everyone-else-on-campus recycled clothing fanatics, the whole retro movement which never goes away yet changes its targets of imitation and afflicts hordes of different troupes simultaneously, or even the weirdo computer types who search out old software and insist on running 30-year-old editors and operating systems—well, you get the point. There's a deep human drive to go backward and worship the old rather than pushing forward to find new frontiers. Here is a game to satisfy both worshippers of everything aged and those who like new ideas.

Figure 1. XInvaders 3D

XInvaders 3D (see Figure 1) from Don Llopis is Space Invaders with a twist, a warp, a new dimension. Yes, that's it, a new dimension: when the aliens come at you, they come *at* you. If you were a fan of the original, you'll be amused to no end by this one. The game is fairly new, so it's not perfect; for example, the graphics don't scale automatically to the size of the window, and there isn't music yet (I suspect a MOD soundtrack just *might* be on the way). Other than that, it's an awesome take on a classic design. Just wait for the filled vector graphs, texture maps, real-time ray tracing. See www.fiu.edu/~dllopi01/xinv3d.htm.

Figure 2. Circus Linux!

Another antique to take a look at is Circus Linux! by Bill Kendrick of New Breed Software. Bill is an interesting fellow who has developed a ton of games for Linux including BoboBot, Defendguin, Gem Drop, ICBM3D & ICBM3D/2, Mad Bomber, 3D Pong, VidSlide and X-Bomber, so I suppose we'll be hearing his name a lot. Circus Linux! is his latest, a Linux remake of the classic (?) Atari 2600 game Circus Atari. You launch circus clowns from a teeter-totter, pop colorful balloons and catch the poor flying fellows, lest they be ill-affected by gravity. Check out http://www.newbreedsoftware.com/circus-linux/ for clowns, balloons and so much more.

a print of King Tut himself or do you have exciting Egyptian graphics, especially Tut, spiders, flies, and other beasts?]

Tutanchamun is a very boring game—just kidding! Actually, Tutankhamen was one of my favorite games on the old Atari. The Atari's chip set is notoriously difficult to emulate, but fortunately we're spared the effort. David Kastrup brought Tutanchamun to Linux. There's an added bonus: David never actually played the original, so what we have is a new interpretation rather than a direct port, a game where the idea was passed from one person to another and morphed into something new, similar to the way urban legends are thought to form. This one is really cute, with the rough, low-resolution graphics and simple game play that fire up the imagination. The Tutankhamen meme mutates and lives! You can get it from metalab.unc.edu/pub/Linux/games/arcade/tutanchamun-0.1.1.tar.gz.

Linux Journal's technical editor **Jason Kroll** (info@linuxjournal.com) suspects that the ability to create a world in our own image is a more powerful incentive in the development of GNU/Linux than the supposed value of peer status.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Focus on Software

David A. Bandel

Issue #73, May 2000

mcdl, disc-cover, cpuid and more.

The other day, I was working on a client's system and noticed the /usr/src directory filled with every 2.2.*x* kernel ever released. When I asked the administrator about it, he responded he always keeps up with each new stable release just in case there are any security-related problems with the old one. (Okay, I'll buy that.) When I asked him why he didn't just download the patches and apply them, he said he didn't know what to do with them. I'm finding system administrators aren't the only ones who don't understand the patch process, as I've sent instructions to a couple of maintainers of very large packages who didn't supply patches.

Those of you with small disks or slow Internet connections can appreciate the problem of downloading 16MB files each time there's a 2KB change. While I'm sure many of you don't notice the impact on the Internet, it's there. Imagine 10% of the folks running Linux (over one million users, if you believe the latest figure of 10 million plus users) wanting to upgrade at the same time. The 2.2.14.tar.bz2 is over 12MB. A patch file from 2.2.13 to 2.2.14 is 1.4MB. That's nearly an order of magnitude difference, and that is significant in both disk space and bandwidth. This amount of traffic would swamp the kernel.org servers as well as several major routers. If you start with new pristine sources each time you compile, you have to go through and recreate your kernel configuration (**make [menu|x]config**). If you patch up, that rather lengthy step is not needed. Using patch files saves everyone time, so save some bandwidth and patch the code.

mcdl: http://rsd.dyn.mil.wi.us/~rsd/code/mcdl/

mcdl provides a fast, easy way to catalog your CD collection automatically, using a MySQL database. The lookup is performed once over the Internet to CDDB sites. A GUI interface is planned for access/lookups/manual input of data.

This is just the back end, but those of you who can use an SQL server (or an SQL monitor like **xmysql**) should find this useful. It requires the MySQL server, Perl, Net::CDDB module, discid, DBI and MySQL modules.

disc-cover: http://www.liacs.nl/~jvhemert/disc-cover/

The **disc-cover** utility will create a print file to print a CD cover for a jewel case. You can include an image file for use on the front cover. It uses a connection to a CDDB site to get the information for the song titles, but a manual mode is available. It can also print extended information, but you need to control the quantity of extended information manually or the layout will be destroyed. It requires the MySQL server, Perl and the FreeDB module.

cpuid: http://people.qualcomm.com/karn/code/cpuid/

Do you have any systems sitting around in which you're not sure what they have? Maybe you want to compile in support for **mtrr**, but only on systems that support it. This little utility will tell you all that and more. If you have more than a few systems around from different manufacturers or different models from one manufacturer, this utility will probably come in handy. It requires glibc.

fdupes: netdial.caribe.net/~adrian2/programs/fdupes-1.1.tar.gz

fdupes is a program to find and remove duplicate files in the directory provided on the command line (required). You must have write access to the specified directory. It requires glibc.

quotenotifier: http://www.bogus.net/~torh/

This Perl script can be run from **cron** or the command line, and will provide the current price of a stock or alert you when the stock goes above or below thresholds passed to the program. You'll need to know the stock's ticker symbol, since no search facility comes with the script. It requires Perl, the Getopt::Long, LWP::Simple and Mail::Sendmail modules and recommends cron.

freq: http://www.bangmoney.org/projects/freq/

This utility reads the lastlog and displays numbers of logins for all users. This is modifiable in several ways. You can include FTP users with shell account users, if you wish. A good informational tool. It requires Perl.

fphdb: http://www.lehigh.edu/~ajr4/fphdb/

This is a very polished, professional order and estimation program for a printing company. Customer, Product, Paper, Shipping, Order and Job Manager

screens access a MySQL database. It requires a web server with PHP support, MySQL and a web browser.

phpLanParty: http://lanparty.hypermart.net/

Want to host a few rounds of Quake? Or another networked game you'd like to invite friends to play? Well **phpLanParty** will let you announce these games in advance, and players can sign up for them. Makes hosting network games easy. It requires a web server with PHP support, MySQL and a web browser.

GNU Pilot LogBook Pro: ftp://ftp.stampede.org/skibum/

This pilot logbook is designed after the Professional Pilot Logbook with which most pilots will be familiar. What you don't need to do is worry about the totals. This is well-laid-out and has a “Remarks and Endorsements” box, allowing you to enter as much data as you want—unlike the actual book, which is very limiting. Medical information isn't implemented yet, but should be added soon. It requires libgtk, libgdk, libgmodule, libglib, libdl, libXext, libX11, libm, glibc, pilot's license and airplane (last two optional).

David A. Bandel (dbandel@pananix.com) is a Linux/UNIX consultant currently living in the Republic of Panama. He is co-author of Que Special Edition: Using Caldera OpenLinux, and he plans to spend more time writing about Linux while relaxing and enjoying life in the “Crossroads of the World”.

Archive Index Issue Table of Contents

Advanced search

Embedded Linux News Briefs

Rick Lehrbaum

Issue #73, May 2000

The latest in embedded Linux news.

Lineo unveiled a challenge to Microsoft's Windows CE: Embedix PDA. The product will add a Win CE compatibility layer to Linux, allowing developers to port Win CE applications to Linux easily. Lineo's targets are palm computers and embedded systems. (http://www.lineo.com/)

Transmeta disclosed plans for “Mobile Linux”, which will support the highly constrained resources of portable Internet devices and embedded systems. The company plans to offer its Linux enhancements to the Open Source community. (http://www.transmeta.com/)

LinuxDevices.com announced the results of its third Embedded Linux poll, which asked developers to describe an embedded computing application in which they are planning to use Linux. The Poll's results can be viewed at http://www.linuxdevices.com/polls/.

A new whitepaper from **MontaVista Software** reviews the benefits of Linux to embedded applications, discusses the alternatives available and offers a glimpse of what to expect from embedded Linux in the coming year. (http://www.mvista.com/)

Corel Corporation entered into an agreement to acquire up to 30% of start-up OE/ONE.com, a company founded by a former Corel executive that has developed a sub-$500 Linux-based Internet appliance. (http://www.corel.com/)

Lineo, Inc. announced it has begun shipping Embedix Linux 1.0, the company's embedded Linux distribution. Embedix is targeted to x86 and PowerPC-based embedded devices. It requires a minimum of 8MB RAM, 3MB of ROM/Flash memory and is based on the Linux 2.2 kernel. (f ROM/Flash memory and is based on the Linux 2.2 kernel. (http://www.lineo.com/)

Evidencing significant inroads made by Linux within the US government, the National Security Agency (NSA) awarded a contract to **Secure Computing Corporation** to develop a robust, highly secure configuration of Linux. (http://www.securecomputing.com/)

Lineo, Inc. announced a major embedded Linux design win in the set-top box market. The system, to be marketed by Bast, Inc., will go in hotel rooms and apartment buildings. The initial plan is for 50,000 systems priced at $285 US each. (http://www.lineo.com/)

Touch Dynamics announced an open-source project to develop KOSIX, an industry-standard public kiosk terminal operating system based on Linux. KOSIX will offer an open-source alternative to conventional, proprietary kiosk OSes. (http://www.touchdynamics.com/)

An interview with Michael Tiemann, **Red Hat's** Chief Technology Officer, discusses the impact of Red Hat's acquisition of Cygnus on the embedded Linux market and the future of the Cygnus EL/IX Embedded Linux API initiative. (http://www.linuxdevices.com/articles/)

IBM disbanded its Internet division and redirected its resources toward an aggressive campaign to promote Linux. As part of this strategy shift, former Internet division executive Irving Wladawsky-Berger was transferred to the new IBM Linux group. IBM says it will collaborate with the Linux Open Source community and has dedicated a portion of its web site to Linux-related information. (http://www.ibm.com/linux/)

DataViews Corporation, a provider of “human-machine interfaces” (HMIs) for factory automation operator interfaces, announced a Linux version of its high-end HMI software tool, DataViews. The company claims to be the first provider of Linux-based HMI tools. (http://www.dvcorp.com/)

UC Berkeley announced two short courses on real-world applications programming, to be offered this spring in Los Angeles, San Francisco and Boston. The courses will emphasize Linux and are titled “Real-Time Programming for Embedded Systems” and “32-Bit Real-Time Operating Systems with an Emphasis on Linux”. (http://www.berkeley.edu/unex/eng/)

There's a movement afoot to develop Linux-based “programmable logic controller” (PLC) technology. PLCs are commonly used in manufacturing and factory automation control systems. A **Linux-PLC** web site, mailing list and open-source software are being created. (http://www.linuxplc.org/)

FSMLabs released beta version 3.0 of RTLinux for “hard real-time” applications. RTLinux can control machinery while maintaining full Linux compatibility. The new release, based on the latest Linux 2.3 kernel, offers improved performance and supports ports to non-x86 architectures. (http://www.fsmlabs.com/)

Linux received a boost in laboratory and industrial test, measurement and control with the announcement by **National Instruments** of comprehensive Linux support. The company has assembled Linux-based instrumentation and control solutions for VME and VXI-based hardware. (http://www.ni.com/)

Intel began delivering prototypes of Itanium, its new 64-bit CPU (formerly code-named Merced). Sources within Intel said the company will shortly authorize the Trillian group (a team working on Linux for Itanium) to release the Itanium Linux source code to the Linux developer community.

MontaVista Software Inc. released its Hard Hat Net CompactPCI backplane networking package to the GPL Open Source community. The move provides developers with powerful networking options for using Linux and CompactPCI in telecom, telephony, Internet and other embedded applications. (http://www.mvista.com/)

VA Linux Systems introduced SourceForge, a major open-source initiative that provides over 700 open-source development projects with extensive hosting and communication resources. The services are available at no cost to open-source developers. (http://sourceforge.net/)

email: rick@linuxdevices.com

Rick Lehrbaum (rick@linuxdevices.com) co-founded Ampro Computers, Inc. in 1983, where he served for 16 years in the roles of VP of Engineering, President and Executive VP of Strategic Development. In 1992, he formed the PC/104 Consortium and then served as its chairman through January 2000. In October 1999, Rick turned his attention to embedded software, founding his second startup: LinuxDevices.com—“the Embedded Linux Portal”. Rick received his BS and MS degrees in physics from NYU and Northeast Louisiana University, respectively.

Archive Index Issue Table of Contents

Advanced search

The Magritte Factor

Stan Kelly-Bootle

Issue #73, May 2000

Warhol's cans of soup are merely “cans of soup” (limited shelf life?), whereas Magritte's equally realistic rendition of a briar pipe is teasingly titled “This is not a pipe”.

René Magritte (1898-1967), the Belgian pop-surrealist, will outlive the undertalented, over-hyped Andy Warhol (1927-1987) in my book. My sister Doreen always called him Andy Arsehole, and thought she was more worthy of the 15 minutes of fame for this remark than Warhol was for his pretentious 3-hour movie of a sleeping man. “Call me a Palestine [note 1],” she said, “but I dozed off after 10 seconds.”

Warhol's cans of soup are merely “cans of soup” (limited shelf life?), whereas Magritte's equally realistic rendition of a briar pipe is teasingly titled “This is not a pipe”. I've made a copy of this painting called “This is not a copy of `This is not a pipe”'--sure to win next year's Turner Prize. Space precludes a full, emetic account of the family visit to the London Tate Gallery in December, 1999. Suffice to say we all stood gazing at Exhibit A, the infamous “unmade bed”, thinking, “Sh..., we used to *dream* of sleeping so bourgeois comfy.”

The Magritte joke digs deeply into the serious anti-self-referential paradoxes that have threatened the very foundations of logic, mathematics, computer science, and all we hold dear [note 2]. The rash Cretan swore that “all Cretans are liars”. Russell denied entry in the only club that would have had him as a member [note 3]. Gödel proved that “this proposition is unprovable”. Turing asked, “Wanna wait 'til this program halts?” And the ever-elegant Chaitin says, “There's no elegant algorithm for establishing its elegance.”

Briefly, in Chaitin's algorithmic information theory, a program is “elegant” if no smaller program written in the same programming language has the same output for a given input. Rush back to your methodologies, patterns, use-cases,

UML flow charts, include files, class frameworks and source code (if any), you inelegant swine! Unless, of course, you are paid by LOC (lines of code).

Most busy humanist mathematicians, such as Lakatos, Hersh and myself, have other rent-paying grapes to fry.

For example, back in the real (sez who?) world, provably elegant Linux gathers momentum. For those who still ask how to pronounce “Linux”, I checked with several IPO billionaires and was told “two short, equally stressed syllables as in Red Hat”.

The latest Corel incarnation has just arrived in the largest-ever shrink-wrapped box! As a JOLT judge, my lips are sealed and incorruptible, but there's a growing number of Linux candidates in each year's list of nominations, and I have an intuitive urge to support Corel and its Linux-aware WordPerfect. As with the Academy Awards, the number of JOLT Award categories also increases, and we are close to having “Best Linux Spreadsheet by an Albanian Unijamb”.

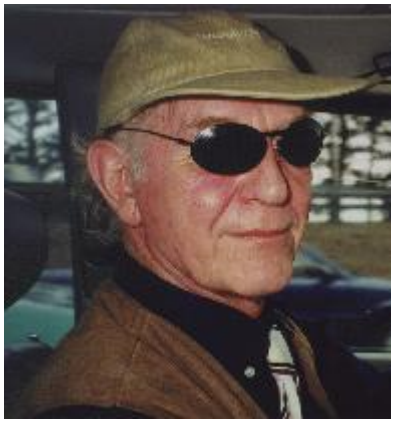
BUT a warning from www.acm.org/technews/articles/2000-2/0128f.html#item2:

“IBM Exec Touts Linux as Key to Net Evolution:

IBM has big plans for Linux, says IBM's Irving Wladawsky-Berger, the visionary who shaped IBM's e-business initiative and now leads the firm's Linux push. In an interview with CNet's Kim Girard, Wladawsky-Berger, who heads IBM's new Next Generation...”

Beware the well-intentioned Big Blue “Kiss of Death!” Especially when it comes from a “next-generation visionary”. Remember OS/2, Taligent, OpenDoc and the emerging Java standards impasse.

Notes



Stan Kelly-Bootle (skb@crl.com) has been computing on and off since his EDSAC I (Cambridge University, UK) days in the 1950s. He has commented on the

unchanging DP scene in many columns (“More than the effin' Parthenon”--Meilir Page-Jones) and books, including *The Computer Contradictionary* (MIT Press) and *UNIX Complete* (Sybex).

Archive Index Issue Table of Contents

Advanced search

Letters

Various

Issue #73, May 2000

LJ readers sound off.

Linksys ProConnect Errata

I made a small mistake in my Linksys ProConnect 4-CPU switch review (March). The current models do not have a console serial mouse port, only a PS/2 port. They do have serial mouse ports for connecting to the computers you wish to control, though.

I apologize for not catching the discrepancy between the manual and the actual switch. That brings the error count in the instruction manual up to two.

I realize the review has already been published, so maybe you could run this in the Letters section of an upcoming issue.

—Ralph Krause rkrause@netperson.net

Painting the Town

I'm still migrating my business, which involves lots of letter and report writing, from NT to Linux, and so found the February story “LaTeX for Secretaries” to be timely. Three things I'd add for anyone contemplating LaTeX for day-to-day office chores are:

One, Lyx is a front end for LaTeX that helped ease the transition for me. I suspect a KDE-flavored version of Lyx, called KLyx, might be good, too.

Two, **word2x** is a terrific little program that converts MS Word documents into ASCII format. Of course, all that formatting is lost in the process, but at least you can get the content. I'm forced to use this far too often because MS Word seems to dominate the non-Linux world.

Three, Leslie Lamport's ancient but still excellent book *LATEX: A Document Preparation System* (1994, Addison Wesley Longman, Inc.) provides, as near as I can tell, everything there is to know about LaTeX in a friendly format.

—Mark Barnes mbarnes@pacifier.com

My Compliments

Consistent technical accuracy and depth of information in the *Linux Journal* is a pleasure to see. I find the depth of the articles and the resource sections better than any book I have purchased on Linux. Of course, there is nothing like reading the man page and /usr/doc/ files. The combination of my subscription for the past three years and the database search on *Linux Journal* Interactive (/) provides a very convenient and efficient way of locating articles to implement software solutions without having to sift through the table of contents of each journal on my shelf. I just wish I had subscribed earlier!

—Kevin Georgison keving@freenet.mb.ca

Another Instant Messaging Client

Your February 2000 issue had a small article about Jabber, an open-source instant messaging system. While I don't want to start our own instant messaging war, I would like to point out that Everybuddy is another alternative. This (http://www.everybuddy.com/) is an open-source client that today supports AIM (AOL Instant Messenger), ICQ, Microsoft and Yahoo.

—David C. Brown, N2RJTn2rjt@localnet.com

Error or Not? You Decide

Just wanted to let you know about an error in the February issue. In the feature article, “Gnome, Its State and Future”, there is an error on page 87. The paragraph entitled “The AbiWord Word Processor” states that “[you] can run the same word processor across UNIX, Win32, BeOS and MacOS.” This is not true. According to the AbiSource web site, on the FAQ page, there is a clear statement under the heading, “Will it work on my computer?”: “Currently, AbiWord does not work on a Macintosh.”

I'd appreciate a clarification in the next issue. I'd also like to say that, although I'm new to Linux, your magazine is one of the few that I read cover-to-cover every month. Congratulations!

—Patrick Beart patrick@WebArchitecture.com

Here is the state of AbiWord for the Macintosh. About three weeks ago, a group of four people got together, engaged in a flurry of activity, and got AbiWord to compile on MacOS. Of course, you cannot do anything with it except admire the created executable. This was a tremendous step forward. Of course, now is when the hard work begins.

—Robert Sievers bob@abisource.com

Letters in Issue 70

Here are my responses to a couple of letters in the February 2000 issue:

Concerning “Ads, Ads, Ads” from Kyle E. Wright, I disagree. I very much like to see the ads in your magazine. So long as you are expanding the magazine and not trading the space with content (which you confirmed), I think it's great. I like to know what's available for my Linux system, and I usually see a product or two each month that I'm interested in. I say, get as many ads as you can—the more money you make, the better your magazine can become.

Concerning “Too Much Red Hat?” from Can Bican, I also disagree. Red Hat is the leader, and as such has more news, more goings-on, etc. I don't mind hearing about them and I certainly don't mind making money in the stock market on Linux companies. Keep the news coming.

—John Dawson jdawson@tcainternet.com

We're Getting Better

You've just ensured my continued subscription.

Thanks for picking up Stan Kelly-Bootle. When I subscribed to *UNIX Review*, Stan's column was the icing on the cake; later, during *UR's* migration to “Performance Computing”, I found Stan's column was the only thing I was reading in that periodical. Now, I'm happy—no, ecstatic—to see that Stan is contributing to *Linux Journal*.

The content of Stan's first *LJ* column seems a little “dumbed down”; perhaps Linux users aren't as educated or sophisticated as UNIX users. Regardless of motives or machinations, I'm very glad to see Stan in one of my favorite magazines again.

—Sean Russell ser@efn.org

Maybe Stan was just getting his feet wet with that first article —Editor

Proprietary Software As God

I do not agree with your magazine's apparent worship of every proprietary software package released for Linux. It is good that Linux is growing in popularity, but you miss the point. Particularly in the February 2000 article in the Forum called “Matlab—A Tool for Doing Numerics”, you disappointed me greatly. GNU Octave is a great program that is a replacement for the proprietary Matlab. It is version 2.*x*, so it has been around a while and works. It can even import Matlab files. The Octave developers deserve our support and thanks for giving of themselves. The Matlab creators do not.

—Pat Mahoney pat13@gmx.de

In the July 1997 issue, we ran an article entitled “Octave: A Free, High-Level Language for Mathematics” by Malcolm Murphy. We have presented many articles on free software and how to develop it. Our column “Focus on Software” is devoted to free software. If you think we worship proprietary software, you obviously haven't read any of Jason Kroll's articles. Creators of both free and proprietary software for Linux deserve our support —Editor

How Many Daemons...

In response to Jason Schumaker's article “The Wide World of Linux” (/article/5381), there were only three daemon women and only *one* of them in latex, a custom outfit of her own choosing. She was a volunteer; the other two women were actresses from a local agency and wore normal red jeans and blouses. All three women enjoyed themselves immensely and expressed great interest in doing this again (it was the volunteer's second appearance; she also did this at COMDEX). Funny how it's always the uninvolved making value judgments about what's sexist and what's not—yet another common defect in human nature. :)

Anyway, since Jason felt compelled to rip on our booth, I figured the least he could do would be to rip on it accurately. Here's some photographic evidence which may also jog his memory for those all-important details: www.freebsd.org/~jkh/lw2000/daemonbabes.jpg and www.freebsd.org/~jkh/lw2000/daemonbabe.jpg.

—Jordan Hubbard jkh@FreeBSD.org

I had intended to write something more here, but I must say the names of your jpg files say it all—daemonbabes, indeed! And what's up with bsdchicks.com? It is true that the uninvolved are often the ones making the value judgements, and that is usually true because the involved don't recognize the sexism in their own actions. —Marjorie Richardson, Editor

The Dæmoness Herself

As one of the BSD girls, the only one in latex, and the one who *uses* the OS and was there as a volunteer, I believe Jason Schumaker ("Going for the Gold", /article/5164) wins the hypocrisy award for assuming none of us were doing this on our own volition, or knew anything about BSD or Linux. ;) And, hey. Coffee substitute and FAQ-answering roles included, it was a *great* show. And I'll include my congrats and a "great job" to Elthia, the *woman* in the dustpuppy outfit.

—Cerene@uclink4.berkeley.edu

For many years, I was a programmer in the oil industry and attended many SEG (Society for Exploration Geophysicists) conventions. At the first ones I attended, most of the booths had pretty women dressed sexily in much the same manner as the dæmon "girls". These women were actresses who were hired for their looks and charm to attract the many men in the industry to the particular booth they worked at. I was offended that women were being used as sex objects and that men were considered stupid enough to fall for such tactics. All these women were happy to have jobs and seemed to be having a good time—this attitude does not change the inherent sexism of the situation. The conventions did not change until enough women became a part of the oil industry to have their voices heard.

BSD's motives may be pure—Mr. Hubbard's letter certainly seems to indicate he feels they are. But looking at the pictures on his site certainly reminds me of the bad old days in the oil industry, and the time and effort women put into changing this sort of attitude. Perhaps using a dæmon guy would help, and costumes not so tight and low-cut.

Jason Schumaker made neither of the assumptions you say he did. I congratulate him for being sensitive to this issue and willing to say so publicly. Congratulations to you on knowing and using BSD! Perhaps when you start thinking of yourself as a woman instead of a girl, you will understand the difference between the dæmon costumes and that of the dustpuppy.

—Marjorie Richardson, Editor

Revisiting the ORB

The Castlewood Orb removable disk drive was reviewed in *LJ*'s December issue. I was intrigued, despite the reported slow speed; I figured there must be a faster SCSI model. In fact, they no longer make the parallel port version you reviewed. It's available as external SCSI or USB, or internal SCSI or ATAPI. I purchased an external SCSI model and I can report the following: I measured

write speed at over 3000KB/sec, and **hdparm** reports reading speed at 8.3MB/sec. Being SCSI, it is trivial to create a Linux-native file system on it and mount it. All and all, I'm very satisfied with my Orb 2.2.

—Peter S. Galbraith GalbraithP@dfo-mpo.gc.ca

I Want My Quicken

You know, I'd *love* to be using Linux full-time as my home desktop OS, but until it can run Quicken and TurboTax, it's a dead issue. In the meantime, it runs great as my home server...

—Jerry Mooth jerrymoo@yahoo.com

Archive Index Issue Table of Contents

Advanced search

Re: April 2000 Issue, Number 72, Letters column, "In Linux We Trust"

Laurie Dare's letter on the Linux Open Source Expo and Conference Program in Sydney (Letters, April 2000) struck a chord with me, and I decided to answer the letter.

First of all, I will mention that this was my third trip to Australia in the past twelve months, where I have given talks at much lower-priced venues, such as CALU and AUUG, as well as free talks at various user groups in Australia.

The $1350 AUS price quoted by Laurie in the letter was the price for both the Linux conference and the ASPCON conference. The Linux conference by itself was $695 AUS, with a Government/Academic discount of 10% off that OR an active LUG member being able to get in for $595 AUS. The dinner price was correct as stated at $125, but again I would like to point out for our non-Australian readers that this was $125 AUS, and not 125 USD. Since USD were trading for 1.82 of AUSD that day, it means the dinner really cost about 68 USD, not an unreasonable price for a dinner in Sydney.

Nevertheless, I was a bit upset to hear that the dinner price was even that high, and I spoke to the organizers about it. They assured me that the dinner was a “break-even” event. I can imagine that, since the Sydney Linux User's Group tried to rent a room for a four-hour meeting in the same facility, and were charged $500 AUS (274 USD) for the privilege of having a room and chairs. This $500 AUS charge was picked up by LinuxCare, much to the relief of the SLUG.

Still not completely mollified by the explanation, I proceeded to put on five more one-hour talks over the next three days of the exhibition, at the small theaters in the Red Hat, Linux Care and SuSE booths. These talks were free of charge, and were well received by the people who attended. Other “luminaries” such as Bob Young, Dr. Andrew Tridgell, Paul “Rusty” Russell and others talked in these theaters as well.

I believe that the Linux community will have to realize that as Linux events get bigger the facilities that can handle them get fewer. You can not have as visible a presentation to 2000 people in a room that has eight-foot ceilings the way you can with a room that has twenty-foot ceilings. And while you have to have large spaces for keynotes such as Linus' keynotes (which tend to draw 6000 or more people), you also need to have smaller rooms for sessions. These types of facilities are hard to find, expensive to build and maintain, and expensive to rent. Nor can an

overhead projector or inexpensive LCD projector do the job of a professional LCD projector (which can cost $500. USD per DAY to rent). Larger events in public places mean the forced use of paid union personnel rather than volunteers.

The exhibition space that was chosen (Darling Harbor's Sydney Exhibition Center) is probably one of the most expensive in Australia. On the other hand, it's proximity to one of the largest population and commerce centers in Australia allowed for a huge number of people to see and find out about Linux. The number of “newbies” that stopped in the exhibition to find out about our operating system was exhilarating.

Still, Laurie has a point, and we should address it. In the beginning of the Linux movement, there were only a few people interested in it enough to go to a conference/show, and those people could fit in one, two or three conferences around the world. “All” of the people were technical, and most of these people were happy to have a place to sleep on a couch in an organizers home (Hi Donnie!).

As the Linux interest grew, more and more events started to occur, with more and more people going to each event. These events not only grew in size, but took on three different flavors for three different markets. There is a fourth “market” that will be discussed later.

The three main markets for “Linux Events” are: the 'Introduction to Linux Newbie Exhibitions', the 'Showcase to the Linux Faithful Exhibitions', and the 'Hackers Delight Conferences'.

The 'Introduction to Newbies“ event should be an exhibit only, with a few talks given to educate. The event is paid for by vendors with exhibit space. Entrance is free to visitors. This is typically located next to a Micro$oft Windows event, or an ISP event, or some other event that will attract the Linux illiterate. Hard Core Linux people may want to stay away (unless you wish to help staff a .ORG stand). The Linux Business Expo (associated with Comdex) is a good example of this type of event.

The 'Showcase to the Linux Faithful' event should be mostly paid by vendors, with enough charge to the conference to cover costs of the conference. Newbies and programmers and sys admins who are paid to go to the event by their management are the main attendees, with a deep discount for students and LUG members. Hard Core Linux people will go to get free CDs and T-shirts, and to attend RAVE parties. IDG's Linuxworld is a good example of this type of event.

The 'Hackers Delight' conference is where the venue is a university, seminary or some hotel in off-season (skiing area in summer, beach in winter, summer camp in late fall, etc.) to house the conference. Attendees stay in local homes of other hackers, or in dormitories. Vendors are limited to 10x10 booths (o.k., for some 10x20, but all simple booths, not huge), and their money goes into funding speaker travel. Realize, however, that volume and quality of attendees are still typically

necessary to attract the speakers and vendors attendees would want, so while these events may still be all-volunteer driven, you may find some of them using paid organizers to help with logistics. CALU and the Atlanta Linux Showcase are good examples of this last type of event.

The last ”market“ for Linux people is not a market at all, but will evolve over the next few years, that of the Linux Workshop. Attendance by invitation only, the list compiled by Linux developers. The venue and travel will be funded by vendors. It will be a place where the developers meet to solve hard problems, or to argue at high bandwidth serious decisions. Perhaps this will be co-located with the ”Hackers Delight“ events, but it will also be separate so the developers can concentrate on the issues instead of what talk they are going to give in a day or two.

These markets have not yet firmed up. You will still find some cross-venues as the people who put them on struggle with the definition of their audience. However, with 30 years in the computer business and many, many trade shows and conferences under (and over) my belt, I have a feeling that it will work out this way. And although I have been known to wear a suit and may present at some of the ”newbie“ conferences, I have a feeling that both Laurie and I will feel more comfortable at the 'Hackers Delight' conference.

Warmest regards,

—Jon 'maddog' Hall, Executive Director, Linux International, maddog@valinux.com

Letter to the Editor

I would like to respond to Phil Hughes' comments on Perl in “Watch out for the Snakes” (January 2000).

The Perl motto of “There's more than one way to do it”, while acknowledging the existence of the quick-and-dirty solution, certainly doesn't encourage it. Efficient programming in Perl comes with practice and effort as it does in every language. But very often the quick-and-dirty solution will work in Perl due to the forgiving nature of its interpretor.

Object-oriented programming in Perl is achieved through the use of modules. It is usual to include information about object variables, instantiation and methods in a separate module file. The definitions of objects in this manner is remarkably easy, as is their use within a program. The archive site http://www.cpan.org/ has over a thousand such modules available for download, testimony to Perl's re-usability and object-oriented capabilities.

Finally, I have been an avid Perl programmer for over two years, was able to write working programs in a short while, and have since been kept interested by the breadth of the learning curve. I have never used **sed** or **awk**, only wrote my first shell script a month ago, and thoroughly recommend Perl to anybody.

I have no doubt Python is a useful language in its own manner, and would appreciate hearing more about its features on their own merit, rather than at the expense of Perl. After all, this isn't an election campaign.

—Christopher Dawson, cmd@maths.uq.edu.au

Keep up the good work!

I have to admit it took a while before I found all your excellent writings, Cluetrain and whatnot. Actually, it was your “Linux for Suits: The Real Meaning of Markets” in the February issue of *Linux Journal* that I stumbled over yesterday. “Ah, at last somebody that sees some of the light” easily comes to mind!

As a Suit myself, I just had to move to Linux a year ago (following the motto of “cannot understand unless you get your hands dirty”). Most surprising result (beside being able to compile kernels, understand the kids and run my own Apache server) was the lead to the Open Source Movement and all the thinking people crowding the newsgroups, Slashdot, *Linux Journal* and Cluetrain—easily the biggest and most interesting social experiment ever!

Being an inhabitant of the old world, about 14 hours away from Silicon Valley, the Net is a boon. Not being able to keep mum I moved a few of my own scribblings (more to come..) onto a web page for the heck of it. If you ever have the time it can be found on http://www.thingamy.net/. The last one, “The Shortest Info Age Business Guide Ever” reflects “my humble view” on some of the issues that you covered in your article in the February *Linux Journal*. BTW, a recent experience with the power of the Internet was when I hacked out a short piece on “A Suit's Experience With Linux” for a local Linux site on a Sunday evening (Feb 6th). Somebody linked it to Slashdot and by the time I got up next morning I had 30 personal mails in my box and in excess of 400 comments to the article on Slashdot. Kind of a revelation for somebody used to old fashioned business...loved the feedback!

Thanks again for sharing and best regards,

—Sigurd Rinde, sigurd@rinde.com

Moving to SMP

Michael Keller wrote a very good article concerning Multi-processor OS's, but I'm afraid it's quite apparent the editors let at least one typo through. In his first paragraph, he writes: “Windows 9x users need not bother, since that platform supports only one CPU per host.” I'm sure he must have meant not ”per host“ but “at most”.

—Greg Leach , gleach@lingerie.com

[LJ March 2000] Canon Fodder comment

In an otherwise good & interesting article on brand naming, there's one minor but troubling statement; Doc Searls states in the second paragraph that “...Red Hat Package Manager (written by Caldera, now used by pretty much everybody)...” To the best of my recollection (and I was involved with both distributions at the time) RPM was developed in house by Red Hat - period. It grew out of the old RPP system, and was incorporated into the Caldera beta releases and Caldera Open Desktop system, the first commercial release from Caldera.

Caldera _DID_, at least in the first phase when the distribution was based on & derived from Red Hat 2.0, help fund some of the RPM development, and Caldera personnel probably have made contributions to RPM, it being an open source project. But RPM was not, to the best of my knowledge, written by Caldera.

—Rick Forrister, rickf@crow.jpl.nasa.gov

> *I take full responsibility for this one. I misunderstood a statement made by a Caldera employee. I added the remark to Doc's article in an effort to be fair. However, it was a misjudgement to do so. The issue is not quite black and white: Caldera did help with the project, but perhaps only through funding as you suggest. Marc Ewing desires all the credit he gets for the development of RPM. I apologize.*
> *—Marjorie Richardson, Editor in Chief*

Best of Tech

I noticed a less-than-good answer in Best of Tech Support section of the march issue of *LJ*. Tim Allison (tallison@logicon.com) asked how to “log out” of linux, and talked about using shutdown(8). Marc replied with an answer that will leave Tim sitting at a linux login prompt, not getting back to LILO. The answer you should have printed was: switch to a text console, and press ctrl+alt+delete. Wait for the machine to reboot. (That's my interpretation of what the guy wanted.) The other way to accomplish this is to have a set-uid program which runs shutdown -r now, such as the following:

```
// I wrote this a while ago, but I don't use it anymore.  (why would I
// to reboot? :)  This code is in the public domain.
#include <stdio.h>
#include <unistd.h>

int main(){
   // printf("UID == %d\n", getuid() );
   // printf("EUID == %d\n", geteuid() );
   setuid(0);
   // printf("UID = %d\n", getuid() );
   execl("/sbin/shutdown", "shutdown", "-rf", "now", NULL);
   perror("Uh oh, can't exec /sbin/shutdown!");
```

```
    return 1;
}
```

You can make this executable only by a certain group if you want to restrict access! OTOH, he obviously trusts the machine's users to be non-malicious, if not clueful :)
Hopefully, one of you will pass some wisdom along to Tim about the three-finger-salute method of shutdown.

Also in BOTS this March, Marc mentions that SMP is supported in 2.2.x kernels, and implies that it wasn't supported before 2.2.x. This is irrelevant, since anyone who doesn't have linux yet will be getting a 2.2 kernel at this point, but it is not true. 2.0.x supported SMP. (2.2 made in an option in the *config stuff, 2.0 you had to edit the Makefile.)

I've been meaning to mail *LJ* about Marc's answers for a while, since he often seems to dash off an answer without thinking about what is being asked. (You did well this month, Marc. :) Previous months have had worse errors and wrong answers that probably didn't help the person who asked :(I could dig out some examples, but I've got some sympathy for you since I don't have time to dig through my stack of *LJ*s... :) I think it would be a good idea to have someone look over the answers given to make sure they are in the right direction at least. It wouldn't take long, just 15 minutes or half an hour to make sure that the answer doesn't have any silly mistakes and that it matches the question. Not to specifically dump on you, Marc, but your answers have had more than your share of bugs. I think it's great that anyone takes the time to answer questions at all, but a wrong answer can confuse people more.

Since I'm emailing you anyway, I'm wondering why you wrote March's Take Command column. I've never used shar or uuencode to send files by e-mail, since everyone uses mail user agents which know about MIME attachments. I just attach the file and mutt or pine takes care of it for me. Your suggestions are applicable to usenet postings, where attachments aren't an option. It seems to me that telling people they need to use shar archives or uuencoding to send files by e-mail might scare them away from trying to use anything other than hotmail (ugh).

Anyway, I hope my suggestions help make *LJ* better :)

—Peter Cordes , peter@llama.nslug.ns.ca

The alienation of the Linux consumer...
I am not to proud of the way that *Linux Journal*, you sister online mag - Linux Gazette, and you competitors have alienated the Linux consumer. There are many, many new product out there for the consumer to get there grubby hands on and proclaim to the world that Linux is here to stay. Yet since I placed my subscription with you all I have seen very few reviews of true consumer product, heard any of the latest breaking news of consumer needs, or truly as much about the consumer market as there should be.

It is true that you last issue brought up a Linux power MP3 Juke Box and I commend that gentleman for his ingenuity. And I have to say that I love these type of articles.

My problem is that there is no true mention of such things as graphics and sound card comparison, new game titles (Heavy Gear, Myth II, & heretic II), an intro to the new crop of Mail order (online and catalog) distributors, etc.

I do like the things that i have seen for the business end of the market but if Linux wants to grow the consumer market needs a carrot now and again. Sure Red Hat, Caldera, SuSE, and VA have stirred up a lot of action but there has to be the consumer base after the business Wagontrain Makes camp and the dust settles.

Thank you for your time.

—Matthew James, Mdjami0409@aol.com

France's later move to Linux ...

In *LJ* #71 (March'2000 issue), Dr Giovanni Orlando wrote : “France moved to Linux some months after the rest of Europe”. What's that ?

When I was still a student, in late 1991, i heard a teacher speaking of the various Unixes you could install onto your PC if you wanted to exercise at home, instead of staying in the classes (the university was all-SunSparc'd, with the exception of a dozen [MS-DOS] PC's and Macs for spreadsheets and those who were reluctant to LaTeX). This teacher said Minix was too minimalistic, Linux would exploit the newest 386 hardware but was still unusable and 386BSD the right choice. So the existence of Linux was known in France at its very first development stages. Some French developpers even contributed then.

In 1995, a serious IT professional magazine wrote an article about Linux and its seriousness. All followed. At the very same time (within the week), i discovered that O'Reilly books (Running Linux, Linux Network Administrator's Guide) had already been translated into French, that some websites had been active into mirroring TSX-11 and Sunsite for more than two years then, translating some HOWTOS and other LDP parts into French, and producing their own French documentation. In 1995 still, there were at least two French Slackware-based distribution that were localized (support for French keyboards and ISO-Latin1 in the kernel and any tools, including GNU- and X-Emacs). The editor of one of these having later been chosen by Red Hat for localizing Red Hat Linux 5.x. Finally, still in 1995, many French ISP were using linux to run their servers and routers (others used FreeBSD or commercial Unixes, one even used NeXTStep). At that time, English-or-French-written books on Linux began to blossom in France from French, Anglo-Saxon or German authors (even German books, once translated into English, were almost instantly available in specialized bookstores).

If you read older *LJ* issues, you can see that some French people authored articles on Linux-based application. Famous examples are those describing the use of Linux in Lectra Systems's tayloring machines, and the design of a new GUI for that purpose.

Maybe Linux trade generally did not catch early in France. There are several reasons for this. First, Linux was long perceived as a way to have Linux for free at home, for training purposes, or to avoid learning Windows for Unix-people. So few people would buy it. If they ever wanted to buy something different, techies would then choose BeOS or OS/2 Warp. Second, buying documentation that could be downloaded for free was not appealing. And third, in 1995, i could read all issues of *LJ* in two neighbour public libraries, and various books about Linux there, for free. The CD-ROM's coming with the books could be lended by these libraries for up to two weeks, so why should we have bought anything? Linux existed in France, but there was no way to measure that.

Another reason why Dr Orlando states France came later to Linux is that French people would not consider an Italian hi-tech company as a source for quality, and so would not buy from it, except for tight budget reasons. I know this is as stupid as a US citizen vaccinating against paludism before coming to France ; you are more likely to get paludism in Florida or New York than in Paris, where the climate is too mild. The same way, French people stick to the old stereotypes, represented by Fiat's cheap low-quality motorcars, or Olivetti's once dubious PC's, and Mercedes's expensive long-life cars or SAP's powerful software. This is stupid because Italy (which has always produced quality products along to cheap ones) also has quality motorcars (Ferrari, Maserati, ...) and well-known Linux masters, while Germany's GM branch (Opel) has become in the last decade a company that produces cars worst than Fiat's, with a quality only suited for the American market. Yet sereotypes are still there and French people would spontaneously trust any training coming from german SuSE rather than from italian FTLinuxCourse.

In the last five years, i have seen all the major american, british, german and french distributions available in France, Red Hat, SuSE and soon Corel being sold localized under their original brand names, and mountains of documentation about Linux. Maybe that didn't measure as money, but money measures nothing about using GPL'd software and docs anyway.

Only dedicated magazines appeared lately, as if the press industry had waited for our politicians' new interest in free software, a consequence of the NSA reconversion into industrial spying in the benefit of US industries leaders and at the expense of US allies industries and US small companies, of Microsoft unfair practices as revealed by the recent trials, and of the supposed placement of NSA agents inside Microsoft, suddenly making all closed-source US software highly suspect).

I thought a *dottore* would think enough to avoid stating something that, by essence, cannot be proven. It is impossible to know which country

came sooner or later to Linux in Western Europe, because you can find contributors, users and evangelists in these countries from the very first days, and because there simply is *no* way to measure that.

—Arnaud alnoken@mail.dotcom.fr

Will not renew my subscription; remove me from your database

My subscription to _*Linux Journal*_ has come due this month. I will not be renewing my subscription.

I have subscribed to _*Linux Journal*_ for many years. However, I have come to realize that _*Linux Journal*_ is, for the most part, not helping the Free Software Movement that inspired the creation of free operating systems.

First, the magazine is quite clearly about an entire operating system, and calls this operating system “Linux” in its title and its articles. Yet, the operating system which many people call “Linux” is not really Linux. Linux is actually the kernel, one of the important components of the system. The system as a whole is basically the GNU operating system. The programmers of the GNU Project are its principal developers, but when people call the system “Linux”, the GNU Project gets very little of the credit for this. (More on this issue is available at: http://www.gnu.org/gnu/linux-and-gnu.html).

Second, *Linux Journal* accepts advertising and frequently promotes (through reviews and announcements) proprietary software for GNU/Linux systems. GNU/Linux thrived on the ideals of the Free Software Movement, which promotes making an ethical choice for freedom, by using only free software. Proprietary software companies are enticing us to give up that freedom and use proprietary software on top of our free operating systems. To maintain our freedom and to continue supporting the development of free software, we must resist the urge to run proprietary software on our free systems. I cannot in good conscience, subscribe to a magazine that does the opposite—promotion of proprietary software.

I encourage _*Linux Journal*_ to change its name to _GNU/*Linux Journal*_, to stop accepting advertising from proprietary software companies, and to never publish articles that are reviews or announcements for proprietary software.

I will happily resubscribe once these changes are made. For now, my personal commitment to the Free Software Movement demands that I no longer fund your magazine.

—Bradley M. Kuhn, bkuhn@ebb.org

Viewable Source != Open Source

Suddenly, it's 1984, again. That was about when the developer community figured out that access to merely _view_ source code was not enough, and they need not put up with the disadvantages of proprietary licences. Proprietary code is fine for those who like it, but might suddenly become unavailable for further improvement and adaptation if (say) the owner withdraws the product, changes business models, or goes bankrupt.

I say it feels like 1984 because of David Penn's 3-Mar-2000 article, “Tripwire Opens Up 'Best of Breed' Security Tool” (http://www2.linuxjournal.com/articles/business/034.html). Penn reports that Tripwire, Inc. will be “providing source code for its flagship product, as opposed to merely open sourcing older versions....”

However, “providing source code” _isn't_ open sourcing. Am I missing something, or isn't this free / open-source world's key, fundamental difference? Did I just dream the last sixteen years? Did, say, Sun Microsystems's SCSL suddenly become an “open source” licence, merely because it makes covered source code open for inspection?

In fact, it is clear that Tripwire, Inc. remains under the (mistaken) impression that viewable source = open source: Its FAQ (http://www.tripwire.org/faq.html) states that “Tripwire, Inc. has had the advantage of distributing an open source product in the market for 8 years”. This refers to “Tripwire ASR”, the viewable-source variant of the company's product. Which, of course, is not open source, and never was.

Examples abound, actually, of the open-source community going to considerable lengths to replace proprietary, viewable-source software, to gain the advantages of genuine open source. The canonical example would be BSD Unix and its progeny. In the security field, GNU Privacy Guard is replacing proprietary PGP, OpenSSH is replacing SSH, _and_ Rami Lehti's GPL'd AIDE package (Advanced Intrusion Detection Environment, http://www.cs.tut.fi/~rammer/aide.html) is making Tripwire obsolete. Tripwire, Inc.'s confusion about licencing is understandable—and no

doubt genuine: They're very late to the party, are considering joining, and misunderstand the ground rules. But it's a little less easy to understand how _*LJ*_ could repeat the company's claims so uncritically.

—Rick Moen , rick@linuxmafia.com

---

tkballistic

I wanted to thank David Bandel for mentioning tkballistic in your Focus on Software column in the Febraury 2000 *Linux Journal*. This is the first time I have ever seen a shooting-related program written up in many years of reading computer magazines. As both a Linux and shooting enthusiast, I've been looking for a ballistics program that runs on Linux. This is something I'll let me online shooting buddies know about.

Thanks again.

—Dave Markowitz, dsmjd@erols.com

---

correction to your Corel Photo-Paint story

Stephen:

In your C|Net article on Corel's release of Photo-Paint for Linux, (http://news.cnet.com/news/0-1003-200-1569948.html) you mentioned Gimp and Adobe as Corel's most likely competitors. This isn't exactly true. First, Gimp has no marketing or business structure. Not even a non-profit. So, although its a terrific program, it lacks the exposure that a commercial application can get. In the long run, this may hurt it.

(I actually toyed with the idea of trying to form a non-profit or even a for-profit to keep Gimp a strong product, but coming up with a business model for this type of application is difficult. Its not likely selling the OS, where service and support can bring in significant income.)

Adobe's move into Linux is limited, so far, to its PDF and word processing tools. Its not, as far as I know, doing anything about porting is graphics or layout applications (though Frame is probably considered a layout too by many). Corel's not really competing with Adobe in graphics on Linux yet.

Mediascape (www.mediascape.com) is about ready to launch its vector based ArtStream for Linux next month. This will be the first entry into the Linux layout tools market. Not long after that, Deneba (www.denebe.com) is expected to launch their Linux version of Canvas 7, a popular Mac image editing tool with vector, layout, and Web development features. These would be Corel's main competitors in the vector graphics arena. Gimp remains a competitor in the raster-based image editing front, but the lack of prepress support and and organizational structure could eventually become a problem.

—Michael J. Hammel, mjhammel@graphics-muse.org

Proprietary Linux Training

Sair Linux Training NO
Caldera Training NO

how come i can't just buy the same materials Sair or Caldera might use in their $2000/week training courses (the binders and CDs, etcetera)? proprietary information. not to be resold, redistributed or copied. that's their very own, very peculiar trademarked way of teaching an open source OS. Phththt! to get some respect in the Linux community you just need the most comprehensive, most difficult certification test not the most expensive training classes. open software. openly available training materials.

when the Free Software Foundation comes up with certification tests as good as their free compilers, i will pay them a yearly continuing education fee and pray every day the rest go belly-up.

—Troy Anderson, satanders@yahoo.com

---

*Vatican City *is* a country (LJ April 2000, Pg 32)*
Peter,

Good morning!

Hate to point this out, but the Vatican City is a country.

It has it's own stamps, passports, police, army, TLD, oh, and a seat at the United Nations.

Europe (as I guess you're American) has a slew of small city states spread all around. Sometimes it is difficult for an outsider to know if they are seperate, but as the man says, if you pay taxes to them and not their larger next door neighbours, you know they're seperate

Examples include, but are not limited to:- Liechtenstein (next door to Germany)
Monaco (south of France)
Andora (between France and Spain)
Isle of Man (between England and Ireland)
Channel Islands (actually a set of countries - between England and France)

Hopefully a small errata can get into the next issue for this!

—Tom Bourke, bourke_tm@mailhub3.davork.com

Letter in LJ
Sent to:scottbomb@hotmail.com

Scott,

I just read your letter in the latest issue of *Linux Journal* and I'm wondering if you've investigated any Linux Users Groups in your area? I think you would be able to find some people who could show you some of the advantages (and disadvantages) to Linux. I started a group in my area two years ago and I am surprised at the range of expertise (not only in Linux) within our group. Most of us work with multiple operating systems on a daily basis and can provide some “real world” comparisons. If you get a chance, You should check into a group in your area. Check out: http://www.redhat.com/apps/community/index.html#lug

Good luck!

—Kim Henderson , kim@aerofil.com

Linux Journal April 2000.

I find myself reading with great disgust your reply to the (possibly trollish) letter published in the April 2000 edition of *Linux Journal*. In his letter, Scott Moore writes:

It's touted as a “free” OS (in other words, you can get it without paying a cent) but it's obviously not.

He continues to discuss his inability to find it anywhere for download, and the best reply you can muster is to point him to a download site. Frankly, I'm ashamed. Not only to you pander to his delusion but you further propogate a myth that's circulating more and more through inaccurate press and word of mouth. Repeat after me:

Free as in speech, not free as in beer.
Free as in speech, not free as in beer.
Free as in speech, not free as in beer.

Please, please, for the sake of your readers and the people currently looking into Linux, set this record straight in your magazine - free software is about FREEDOM, not PRICE. The fact that the price happens to be zero or near-zero is a nicety and most definitely not the whole enchilada.

Perhaps you felt a more thorough reply was not required due to the obvious rant-factor (perhaps my letter will rate similarly on that scale, in the opposite direction?), but this is (IMNSHO) one of the worst mistruths circulating in the minds of the general public.

—Dave Baker , dave@dsb3.com

Linux and IBM PowerPCs

As a regular reader of the *Linux Journal* and a happy IBM RS/6000 owner running Linux, I was quite satisfied with the March issue, and its Linux and IBM PowerPC article. I have a few comments on it, though.

First: There are availble more than one Linux Distribution in the world that will run on your IBM RS/6000, allthough Yellow Dog is the only one that is supported by IBM. The LinuxPCC distribution has a lot of the core Linux/PPC developpers in the “stable”, and has maybe the best unofficial support on Linux/PPC via its mailing lists. SuSE's upcoming release 6.3 for PowerPC will run on pretty much the same workstation models as Yellow Dog. Same goes for Debian and its coming 2.2 distro. I have successfully installed Yellow Dog, LinuxPPC, the SuSE 6.3 beta, and Debian Potato on my 7248 system without too much hazzle. There are availble step-by-step installation instructions for several of the distributions mentioned.

Second: The main reason that this is possible is of course that the Linux kernel is the same in all the distributions, so that if one distro can run on a special model, there won't be too long till all of the others can do the same.

Third: IDE based IBM workstations and laptops have some trouble running with the stable release of Linux. Working sets on the mailing lists use patched kernels from the unstable 2.3 series. Hopefully we'll have beautiful support for this machines when the stable 2.4 series arrives. Until then it's quite possible to get your IDE machine running, but prepare for some hours of work.

Fourth: The main reason that older IBM PRePs are not officially supported by Yellow Dog is quite possibly the lack of a real working X server to this kind of hardware. Because of this, the PRePs won't reach it to the desktop unless running as a server, connected to a good X terminal. Hopefully, we'll have working X servers in the future, supporting this kind of hardware.

Last: A good list of working hardware for Linux on PPC, and pointers to installation instructions for several kind of machines can be found at http://www.linuxppc.org/hardware The LinuxPPC distribution can be found at http://www.linuxppc.org , and is commercially sold from http://www.linuxppc.com Unofficial installation support can be found at http://lists.linuxppc.org (the url in the article has a typo) The SuSE 6.3/PPC beta can be found at http://www.suse.com/ppc Debian Potato/PPC information can be found at http://www.debian.org/~porter and in the unstable tree on a debian ftp mirror

Regards, Ingvar (Who is writing this on his GNU/Linux-running 7248-133 while listening to mp3 on it's built in sound adapter, running GNOME, and sending this mail via it's built in ehternet adapter.)

—Ingvar Hagelund; UiO, ingvar@unik.no

Article on NetMax in Issue #72

Greetings.

A comment from the trenches on your review of NetMax WebServer.

I had a less-than-stellar experience with the NetMax Professional version of this product in an [granted] chaotic environment of 11 Macs (of various ages), 2 NT workstations and 2 W95 boxes. Though perhaps adequate for stable and non-disruptive environments, it proved untennable in this one. Small configuration and hardware problems proved enormously difficult to track down, and we are reverting to a vanilla Linux distribution where config files are under our own control. Where things are behaving again.

Users should be aware that the CGI scripts that control all the service daemons so nicely presented in the web interface are completely closed-source. Moreover, many of the familiar /etc/rc.d config files are no longer relevant. Consequently anyone who relies on being able to solve network configuration problems by hand-editing config files may be frustrated. As I was.

Though the tech support responsiveness from NetMax was great, ultimately the fundamental problems of a closed system began to feel too much like my battles with M$.

A few simple examples:

- would you like to use WVDIAL with PPP? no can do. Would you like to set the debugging option CHAT to diagnose login failure? sorry.
- would you like to control which interfaces SAMBA and/or ATALKD are configured to run on? too bad.

—David Updegraff , dave@toimi.com

Desktop "wars"

First, nice work. I've been a subscriber for a while and I'll continue. That's a vote of support where it counts.

The main reason I'm writing is to send a note about the Gnome/KDE conundrum [including Phil's editorial in support of KDE]. Some of us don't want the windows desktop. If I wanted a “task bar” and a start menu, I'll just boot into NT [actually, regardless of whether I want them or not, I get them]. These two are not the only game in town. The series on “Artist's desktops” is a step in the right direction, but there's more to X than E or an M$ windows wannabe.

I've run 9wm, wmx, and am currently using ctwm. I like ctwm because it gives me the virtual desktops I need, sound feedback on some events [via rplayd], and allows me to configure keyboard/mouse click shortcuts as I see fit. Of course, I need to read and think and take the time to change things [an ongoing process], but I like it this way. I've always thought that twm had the right design philosophy- and ctwm fits right in there.

Now you may think I'm a hopeless UNIX geek- but even outside of my personal preferences, it's pretty easy to see how one-size-fits-all doesn't work in sneakers, and it doesn't work in desktop metaphors either.

I'm not interested in telling people what to run on their desktops, but to me, Gnome/KDE is a step backwards [toward Redmond] instead of forwards [towards individual customization].

—John Saylor , jsaylor@mediaone.net

Ranting and Raving Back

“Do you wanna know what really pisses me off about Linux?” These are the words spoken by Scott Moore in the April issue of *Linux Journal*. For a moment I thought that the letter must have been written by Bill Gates himself until I continued on and realized that the reason he is so upset is that he is having a hard time downloading a copy.

Is it just me or is this the absolute most rediculous thing you have ever heard? I wonder if Scott even really looked for a place to download the software or if he just touted the MS Party line and concluded that all might Bill has to say is true.

As for the lack of software available for Linux, I suggest that Mr. Moore take a look at freshmeat.net or any of the dozen other sites that I can think of where you can get source code and pre-packaged software for all the major distributions of Linux. My belief is that Mr. Moore is afraid of learning something new and is unwilling to go take more than one step beyond his install shield installer and his start button. Kudos to all of you who contribute to the production of this fine zine and to all the great software developers and systems programmers out there who have made the cult called Linux available for the rest of us who are not afraid to try something different.

—Andrew Armstrong , slipknot@highertech.net

DSL Access

I read with curious interest Jason Schumaker's article in Issue #72 of *Linux Journal*. I think Jason is wrong on several points.

One, you don't need a router for DSL. I directly connected my computer to the DSL modem and it's worked fine. You can hook up a hub to the DSL modem and drive more computers, but you have to get more addresses from the ISP. If you want to go the cheap route and need a router, Linksys makes one with 5 Ethernet ports, works with a DSL modem, and has a 4-port hub for $250. I've seen this router as low as $150 to $180 depending where you go on the internet.

Two, I think what Jason was lamenting about was the situation in his community. I live in Houston Texas, and I've had DSL for about a year (April 1, 1999 was my install date). Southwestern Bell services this area and I'm very happy for the service. SWBell is now offering DSL service with free installation and hardware for a one-year commitment. And you get to choose your internet provider. I ordered my DSL service through my provider, and if I have any problems, I call either SWBell if there's a problem with the line, or my provider if there's another problem. There's no questions on whom to call if the line goes down.

Three, Covad, Northpoint, and Rhythms buy the lines from the local ILEC and sell the service bundled with the DSL line. Many providers, like Internet America, use Northpoint to provide DSL services. Remember these companies are targeting businesses, not regular home users.

SWBell targets both residential and business users, so I think a person would be better off going with DSL from their local ILEC.

I do agree with Jason about Cable Modems. I think the cable modems are a losing proposition because the providers assume Windows stations will be the clients. With the customers in hubs, there will be congestion just like the dial-up modems. The DSL line offers a dedicated pipe. The phone companies are eager to make more money from these DSL lines.

I think right now the decision to make is not based on which service, but rather what area one is in. Down here, DSL is dominating cable modems due to the slow rollout of cable modem service. SWBell had a 6-month head start on their high-speed service ahead of Time-Warner, for example.

Keep up the good work.

—James A. Buckner, jbuckner@texas.net

> *Well, yes, Jason was writing about his area specifically, but the same problems happen in other areas too. As a matter of fact, I have friends who live in Houston (my home town) who have never been able to get DSL service and have resorted to cable modem. So perhaps they like Jason, live just outside the limit for being able to use SW Bell's service. Also, the DSL modem is not a true modem, it is a type of router. —Editor*

Issue 72 - "that's not an uptime....This is an _uptime_."

68% of statistics are made up or wrong. :)

In *LJ* Index for April 2000, it's stated that the longest uptime for a Linux system was 498 days. That's just not right. Ask around and you'll find people who can tell you about their 500+ day uptimes. Mine ended somewhere roughly around 570 days when the system needed to be physically moved. I can't find in any of the popular search engines my final uptime post when that system was shut down, but I know I did post about it.

Here are two from shortly before that.

http://linuxwww.db.erau.edu/mail_archives/server-linux/Sep_97/0096.html
http://linuxwww.db.erau.edu/mail_archives/server-linux/Nov_97/0240.html

I've seen stories from others about multi-year uptimes with Linux. Claiming 498 days as the record is selling Linux way short.

—Jon Lewis, jlewis@lewis.org

At the forge—April 2000

Just got through reading the April 2000 issue....it was great as usual. In particular, the “At the Forge” column by Reuven Lerner was a good beginners article on SQL database design. I have one quibble, however (there is *always* an however, isn't there?): He creates a table (StationLines) that has a column north_to_south, which “is an integer value that counts the number of stops between the beginning of the line and the named station.” Doesn't this make it a pain if, as he asked rhetorically on the first page of the article, “a new station is built between Haifa and Tel Aviv?” You would have to go into the tables and renumber all of the stations that came after the new station. Wouldn't it be better design to use a constant value, in this case maybe milepost number? (Station A is at milepost 0, Station B is at milepost 10.2, Station C is at milepost 45.9; the new Station D is at milepost 15.6...we don't have to renumber anyone, and the SQL query can still pull them out in north to south order. Just a thought...

—John Quentin Heywood, heywood@wcl.american.edu

---

up Front - April 2000

your source for point 14 in *LJ* Index (498 days of the longest uptime on Linux systems) is grossly misinformed. In my own experience on one consulting jobs I had a Linux machine which at the moment we are parting our ways with this customer had already around 650 days of uptime. If they not tore down a building which is housing it (there was such possibility) it may well still be running now. I do not know. 650 days was in my understanding slightly amusing but nothing that remarkable for a server, as opposed to a “hack-box”, with UPS.

In recent discussions on linux-kernel list somebody mentioned his machine which is closing now to 1300 days of uptime. 498 is not even cutting close. It is true that 'uptime' utility counter will wrap around after roughly 498 days but this does not mean that a box stopped. The same way as a car odometer which wrapped around the third time does not make this car brand new. :-)

Michal Jaegermann, michal@harddata.com

—Michal Jaegermann , michal@ellpspace.math.ualberta.ca

Please post in the Letter to the Editor on Linux Journal

May I propose that you post this in the “Letters” section of *Linux Journal*. Thank you for an EXCELLENT magazine.

May I define two concepts? The “open-model” is the concept that ideas from one person/organization may freely pass to another person/organization in order to collaborate (http://www.opensource.org/osd.html). The “closed-model” is the concept that ideas from one person/organization are restricted to those within their circle of control.

History shows solid, ample evidence that both models create competition while the open-model feeds progress and the closed-model stifles progress. I hope not to detract but solidify my position, by providing a few examples. These examples can extend in every facet of life (Government, Education, Society, Art, Economy, etc.). May I detail the value of open-model in the technology facet.

In 1820, Danish physicist Christian Oersted demonstrated electromagnetism by pushing a compass under a live electric wire. This caused its needle to turn from pointing north, as if acted on by a larger magnet. Oersted discovered that an electric current creates a magnetic field. In 1821, Michael Faraday reversed Oersted's experiment. He got a weak current to flow in a wire revolving around a permanent magnet. In other words, a magnetic field caused or induced an electric current to flow in a nearby wire In 1830, Professor Joseph Henry transmitted the first practical electrical signal. Showing that electromagnetism could do more than create current or pick up heavy weights—it could communicate.
In 1837, Samuel Morse invented the first workable telegraph. In 1861, Johann Phillip Reis completed the first non-working telephone. Tantalizingly close to reproducing speech, Reis's instrument conveyed certain sounds, poorly, but no more than that. Alexander Graham Bell, on the other hand, saw telephony as the driving force in his early life and become know as the father of the telephone.

Source: http://www.privateline.com/TelephoneHistory/History1.htm

All of these followed the open-model of collaborating information and NOT keeping it closed.

I believe the closed-model is not bad; however, the open-model is a much better way.

The natural, basic, fundamental, or primal outcome of open-source is progress. Open-source breeds healthy competition, which breeds incremental improvements. In contrast with the closed-model, the open-model does not blindly reinvent the wheel but insightfully works with others and builds upon or corrects what others have already done.

Often I see arguments about which model is better. The open-model advocates have nothing to worry about so why cause a fuss? Are some open-model advocates criticizing closed-source software companies because they are concerned the closed-model will get ahead, become better, blackmail open-source, or prohibit open-source?

Close-source software may very well continue to get ahead of where it currently sits. That is fine. Let them go at their pace and open-source will exceed their pace.

Can closed-source becomes better software? Well, sure. That is their goal and some may argue they have done that. They will with us on their tails

—pushing their movement—providing healthy competition. Yet closed-source companies will compete with some of the best and smartest programmers this world has to offer.

Will closed-source blackmail open-source advocates? They have and I believe they will continue. So what! Truth, over time, will triumph. Perjury will lead to eventual death. We tend to find out the correct details sooner or latter.

Will closed-source companies prohibit open-source? If they are successful, then we have damned ourselves. “Dam: a barrier built across a flow of technological information impeding progress.” This is where we need to legally and prudently fight. The strategy to prohibit or otherwise choke the open-model is a direct attack on the core of the open-model—FREEDOM. This is where the real danger lies. We should be concerned with this threat.

I believe the open-source advocate's focus should not be to waste time, resources, and efforts on criticizing closed-source or the closed-model principle. Where will that get us? If angry people just don't get the open-model concept, then getting them more upset won't help them. Some criticizing has lost (in the argument) the fundamental hallmark of open-source: FREEDOM to collaborate.

The entire fundamental concept of open-source or the open-model is FREEDOM to see, decide, and act.

The open-model leads to rapid success and a change of attitude for some people.

—Valden Longhurst , valden@longhurst.com

Installing SUSE 6.3 on my Toshiba Laptop

The installation was fairly straight forward.

I first changed the BIOS BOOT Priority to FDD > CD-ROM > HDD to enable the system to boot directly from the CD. This then started the installation process which I found to be very easy to use, although I was expecting at some stage to be presented with a list of available packages, missing dependencies and a host of error messages to tell me it didn't what hardware I was using. To my surprise(and joy!) with only a few easy decisions to be made, I found the installation progressing extremely well.

Once I got the system booted, I found that it would get to the login prompt and then Freeze. After some thought (and some swearing and a bit of Jack Daniels) I hit on the idea of the PCMCIA settings(don't ask me why or how) this is when I changed the BIOS settings from Auto-Selected to CardBus/16-bit and hey presto! a working, booting system.

Then came X.

I had initial problems with the Xserver config and had to fish around on the web for some assistance. http://www.cs.utexas.edu/users/kharker/linux-laptop http://www.buzzard.org.uk/toshiba http://sdb.suse.de/sdb/en/html/grimmer_cyber9385.html These three sites gave me a few pointers but I had to resort to using an external monitor in order to get SaX running when I got fed up with hacking XF86Config files manually(I could never get the syncs correct). I would definitely recommend using an external monitor for setting up X on a laptop(especially a Toshiba because they have the 'Hot Switch' FnF5)

After X was up and running, I then realised that the DHCP client was not connecting to my NT server and picking up a valid address. I didn't want t spend too much time with this so I gave the machine a permanent IP address.

I then set up Samba to use the Printer on the NT server using yast. (Easier done than said!)

I then set up Netscape and StarOffice to access my Proxy server and I now have a portable Linux Workstation.

Once this was sorted, I set about installing Sybase 11.9.2. This told me that I needed certain versions of libraries and kernels which were, in fact, older than the installed versions so I installed the packages with the -nodep option. This worked a treat!

I shall soon be getting DHCP working (although this is not a priority) and also I am going to try out my portable HP CD Writer. This should keep me busy.

I have included my XF86Config file and machine configuration below, just in case anyone is interested.

Regards,

—Sean Hackett, seanh@teknosys-corp.co.uk

```
##################################################################

# SaX autogenerated XF86Config file
# This file was generated from the SaX
# Version: 2.8 - sax@suse.de
# Date: Fri Feb 25 20:56:43 GMT 2000
# Xserver:SVGA
# MouseVendor:Unknown
# MouseName:Unknown
# RamDac:207
# Dac8:207
# Dac16:207
```

```
# Dac24:
# Dac32:

Section Files
   RgbPath  /usr/X11R6/lib/X11/rgb
   FontPath /usr/X11R6/lib/X11/fonts/75dpi:unscaled
   FontPath /usr/X11R6/lib/X11/fonts/100dpi:unscaled
   FontPath /usr/X11R6/lib/X11/fonts/Type1
   FontPath /usr/X11R6/lib/X11/fonts/URW
   FontPath /usr/X11R6/lib/X11/fonts/Speedo
   FontPath /usr/X11R6/lib/X11/fonts/misc
   FontPath /usr/X11R6/lib/X11/fonts/75dpi
   FontPath /usr/X11R6/lib/X11/fonts/100dpi
EndSection
Section ServerFlags
AllowMouseOpenFail
EndSection
Section Module
EndSection

# This section is no longer supported
# See a template below
# Section XInput
# EndSection

Section Keyboard
   Protocol Standard
   XkbRules xfree86
   XkbModel pc104
   XkbLayout   gb
EndSection
Section Pointer
   Protocol    PS/2
   Device      /dev/psaux
   SampleRate     60
Emulate3Buttons
   Emulate3Timeout   200
   BaudRate    1200
EndSection
Section Monitor
   Identifier  Primary-Monitor
   VendorName  !!! LCD !!!
   ModelName   XGA 1024X768@60HZ
   HorizSync   31.5-50
   VertRefresh 58-62
Modeline 1600x1000 104.00 1600 1616 1968 2080 1000 1000 1007 1044
Modeline 1280x960 83.20 1280 1296 1552 1664 960 960 967 1003
Modeline 1024x768 66.00 1024 1040 1216 1328 768 768 775 802
Modeline 640x480 25.79 640 656 720 832 480 480 484 501
Modeline 1600x1200 104.00 1600 1616 1968 2080 1200 1200 1207 1253
Modeline 1280x1024 83.20 1280 1296 1552 1664 1024 1024 1031 1070
Modeline 1152x864 74.80 1152 1168 1384 1496 864 864 871 902
Modeline 800x600 40.35 800 816 928 1040 600 600 606 626
```

```
EndSection
Section Device
   Identifier  Primary-Card
   VendorName  !!! GENERIC SERVER SELECTION !!!
   BoardName   TRIDENT
EndSection
Section Screen
   Driver   SVGA
   Device   Primary-Card
   Monitor  Primary-Monitor
   DefaultColorDepth 16
SubSection Display
     Depth  32
     Modes  640x480
EndSubSection
SubSection Display
     Depth  24
     Modes  640x480
EndSubSection
SubSection Display
     Depth  16
     Modes  1024x768
     Virtual   1024 768
EndSubSection
SubSection Display
     Depth  8
     Modes  640x480
EndSubSection
EndSection

Section Screen
   Driver   Accel
   Device   Primary-Card
   Monitor  Primary-Monitor
   DefaultColorDepth 16
SubSection Display
     Depth  32
     Modes  640x480
EndSubSection
SubSection Display
     Depth  24
     Modes  640x480
EndSubSection
SubSection Display
     Depth  16
     Modes  1024x768
     Virtual   1024 768
EndSubSection
SubSection Display
     Depth  8
     Modes  640x480
EndSubSection
EndSection
```

Machine config.

Manufacturer : Toshiba
Model : Satellite 4060XCDT
Hard Disk : 4.3 Gb
Memory : 131072KB
Display Chipset : Trident Cyber 9525 2.5Mb
Network Card : 3COM PCMCIA CardBus 3CCFE575BT
   Domain Controller is an NT SBS machine
   Currently also have a Red Hat 6 server running a development Sybase server

BIOS : ACPI version 7.60
Bios Settings
First Page

DISPLAY
Power On Display = Auto-Selected
LCD Display Stretch = Disabled

BATTERY
Battery Save Mode = Full Power

PERIPHERAL
Pointing Devices = Auto-Selected
Ext Keyboard Fn   = Disabled
USB Legacy Emulation = Disabled
Parallel Port Mode   = ECP
Hard Disk Mode   = Enhanced IDE(Normal)

OTHERS
Power-up Mode   = Boot
CPU Cache   = Enabled
Level 2 Cache   = Enabled
Auto Power On   = Disabled
Alarm volume   = High
System Beep   = Enabled

Next Page

CONFIGURATION
Device Config.   = All Devices

PC CARD
Controller Mode   = CardBus/16-bit

I/O PORTS
Serial   = COM1(3F8H/IRQ4)
Built-in Modem   = COM2(2F8H/IRQ3)
Parallel   = LPT1(378H/IRQ7/CH3)

DRIVES I/O

```
  HDD          = Primary IDE(1F0H/IRQ14)
  CD-ROM          = Secondary IDE(170H/IRQ15)

  DISPLAY
  VGA Segment Address  = C000H

  FLOPPY DISK I/O
  Floppy Disk     = (3F2H/IRQ6/CH2)

  PCI BUS
  PCI BUS         = IRQ11
```

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

upFRONT

Various

Issue #73, May 2000

Stop the Presses, *LJ* Index and more.

STUPID PROGRAMMING TRICKS

Welcome again to Stupid Programming Tricks. May is programming month! Actually, every month is programming month. Apparently we can't call it coding, since we're using C rather than assembly code—which is too bad, since “coder” sounds much more sinister and intelligent than “programmer”. Regardless, lately I've been wondering, am I the only Linux user in the world who didn't come from a UNIX background? The truth is, all this networking bother strikes me as a bit dull. What about normal coding? If Linux is going to sit anywhere other than on servers, it's going to have to be a multimedia OS. You may have noticed all of our Stupid Programming Tricks episodes deal with multimedia. Well, there's a method to my general idiocy. Already we can write scrolltexts, slide graphics over backgrounds, play music, synthesize sounds, fork processes and placate **gcc**. We can also log in as root and ruin our console screens, but that's beside the point.

This month, in honor of programming, we're going to make something neat. Yes, it's time to do something really cool and yet somehow worthy of the title “Stupid Programming Tricks”. To justify such, we must be explicit in doing something with no practical value. Hence, the point. What better way to say “Hello world!” than with a scrolltext? Hark, someone is *senile*—we have already *done* scrolltexts. Grrrr. How about a scrolltext that warps like a slithering snake?

Sine scrollers are an epitomal form of the scrolltext, not quite the pinnacle—for that, we'd need text that wraps in three dimensions, casting shadows and twisting like threads of deoxyribonucleic data—but an epitomal form, nonetheless. We're also going to cheat. Well, not really, but instead of doing clever programming tricks to save processor resources, we'll use a simple

routine, since with SVGALib, it's the only thing we can do. Sine scrolling is a trick in itself, and I haven't found any tricks to make it less processor-intensive, short of adding new functions to SVGALib or lowering the resolution. It works on the 1MHz C64, and it ran on my 60 MIPS box with digital music in the background, so I reckon your 600 MIPS box won't suffer.

In order to make a sine scroller, we start with the makings of normal scrolltext: a physical screen for the actual display, a virtual screen for drawing and a scroll board to hold a graphic of our scrolltext. In our last scrolltext episode, we just copied the scroll board to the virtual screen and the virtual screen to the physical screen, doing all the scrolling in the scroll board and drawing our text as one graphic the width of the screen. This time, every single pixel-wide vertical strip of text will have a different Y coordinate from its neighbor when it gets drawn to the virtual screen from the scroll board. The only way to do this is to copy a 1x8-pixel strip, 320 times every refresh! That's about 19,200 function calls per second, even at a 60Hz refresh rate. Well, with processors that can handle hundreds of millions of instructions per second, it's not a big deal. My old box, which ran at 60 bogomips before it died last summer, could do a sine scroller with a really big 256-color font, star fields and raster bars in 640x480x256 with digital audio in the background. Remind me to dig that code off the hard drive and cover it some time. Today, we'll just use the standard 8x8 font in one color, to get the hang of the whole sine-scrolling thing.

Sine functions are beautiful. You can use them for anything, literally. A sine function has amplitude, phase, period and shift, and you can play tricks with these. For example, you can plug a sine equation into the amplitude, phase, period or shift on your first sine equation. These techniques are useful not only for making cool patterns; audio synthesis techniques rely heavily on plugging sines into sines. For our scrolltext, we could make a sine equation with a period of, say, 320, an amplitude of 90, and shift it to the exact middle of the screen. However, this would result in an unchanging sine pattern; that is, rather than slithering like a snake, the sine pattern would hold still like a serpentine pipe while the letters contorted through. It looks cool enough this way, but there's no sense of motion. To add motion, we play games with the phase; i.e., we keep cycling the phase so that the sine appears waving, up and down, like a snake.

Now, how steep do you want the sine? Well, that's a function of amplitude and period. Generally, we already have an idea of what amplitude we want (usually either full-screen or just bouncing along the bottom), so we play with our period. If you want a scrolltext that squeezes together and then stretches out, you can use a sine function to modulate the period. Now, we could also plug a sine into the amplitude, to move between flat-lining and squiggling, although it's not so useful.

Finally, you can also plug a sine into the shift in order to give a lot more variety to the scroller as a whole. The result actually looks quite neat, it's rather complex, yet visibly based on sines. There's an aesthetic quality here to be appreciated, the interplay between simplicity and complexity. In honor of sine waves, let's also grind our processors a bit by using sines to cycle the colors of our text; it'll look cool.

Now it's time for the code. Remember, it's just the same simple procedure as in our last scrolltext episode, only now we're using some sine equations to determine the *y* value for where our text gets placed, and a loop to draw out the text one 1x8 strip at a time. It's simple conceptually, although if you don't get it, try typing it in rather than downloading it and you'll probably understand exactly what's going on by the time you are done. Sines are some of the most useful things in the world, so even though we're using them for something silly, they have infinite practical use, and I hope this inspires some clever ideas for you!

—Jason Kroll

Listing. Fun With Sines

```
// gcc -Wall -O2 sine.c -lvgagl -lvga -lm
#include <vga.h> /* vgalib */
#include <vgagl.h> /* advanced */
#include <math.h> /* sines! */
#include <stdlib.h> /* malloc */
#define VGAMODE G320x200x256
GraphicsContext *physical_screen;
GraphicsContext *virtual_screen;
GraphicsContext *scroll_board;
int main(void)
{
  double a, b, c, d; /* amp, phase, period, shift */
  double aa,bb,cc,dd; /* random values for fun */
  short int p_pos, t_pos; /* pixel & text */
  short int x,y,z; /* y coord, x & z counters */
  char key; /* to wait for keypress */
12345678901234567890123456789012345678901234567890
/* The following text between quotes should be all
   one line. Ignore wrapping */
  char text[] = ".................................Hello happy world and welcome to another episode
  double textl=sizeof(text); /* text length */
  vga_init(); /* here begin standard inits */
  vga_setmode(VGAMODE); /* we'll allocate our */
  gl_setcontextvga(VGAMODE); /* graphics areas */
  physical_screen=gl_allocatecontext();
  gl_getcontext(physical_screen);
  gl_setcontextvgavirtual(VGAMODE);
  virtual_screen=gl_allocatecontext();
  gl_getcontext(virtual_screen);
  gl_clearscreen(0); /* better clear the screen */
  scroll_board = malloc( (WIDTH/8+1)*8*8*BYTESPERPIXEL);
  gl_setcontextvirtual(WIDTH+8,8,BYTESPERPIXEL,8,scroll_board);
  scroll_board = gl_allocatecontext(); /* ready */
  gl_getcontext(scroll_board);
  gl_setwritemode(FONT_COMPRESSED);
  gl_setfont(8,8,gl_font8x8);
  gl_setfontcolors(0,1);
  gl_setpalettecolor(1,63,13,24);
  srand(time(NULL));
  a=b=c=d=x=y=0; /* amp, phase, period, shift */
  t_pos=p_pos=0; /* text & pixel position */
```

```
    aa=rand()%64+16.0; bb=rand()%64+16.0;
    cc=rand()%64+16.0; dd=rand()%128+128.0;
    gl_setcontext(virtual_screen);
    /* main loop */
    for (key=0; key==0; key=vga_getkey()) {
      p_pos += 2; /* adjust speed here */
      /* this redraws the text at each new letter */
      while (p_pos > 8) {
        gl_setcontext(scroll_board);
        gl_writen(0,0,WIDTH/8, &text[t_pos]);
        t_pos++; /* advance text */
        p_pos-=8; /* reset p_pos */
        if (t_pos >= textl)
           t_pos=0; /* reset t_pos */
        gl_setcontext(virtual_screen);
      }
      /* These equations produce readable text
       * but please experiment to witness the
       * potential of illegible sine scrolling
       */
      x+=1; /* a counter for more phase shift */
      for (z=0; z<320; z++) {
        a = 24*sin((z+x)/aa)+24;
        b = 64*sin((x)/bb)+32;
        c = cc*sin((z+x)/cc)+128;
        d = (32-a)*sin((x-z)/dd)+124-a;
        y = a*sin((z+b)/c)+d; /* standard format */
        gl_copyboxfromcontext(scroll_board, z+p_pos,
           0, 1, 8, z, y);
      }
      gl_setpalettecolor(1,31*sin(t_pos/2.0)+32,
         31*sin(t_pos/4.0)+32,
         31*sin(t_pos/8.0)+32);
      gl_copyscreen(physical_screen); /* update */
      gl_clearscreen(0); /* otherwise it smears */
      vga_waitretrace(); /* hold still a*/
    }
    return 0; /* on principle ;) */
  }
```

LJ INDEX—MAY 2000

1. Total number of patented applications listed by the U.S. Patent & Trademark Office as of June 30, 1999: **2,090,902**
2. Percentage of those patents with foreign origins: **43%**
3. Position of Japan among foreign countries holding U.S. patents: **#1**
4. Percentage of foreign-origin patents held by Japan: **41%**
5. Position of IBM among U.S. patent holders: **#1**
6. Number of patents held by IBM: **20,725**
7. Position of Canon among U.S. patent holders: **#2**
8. Number of patents held by Canon: **18,043**
9. Number of Japanese companies among the top ten U.S. patent holders: **6**
10. Number of patents held by Microsoft: **1,167**
11. Number of patents held by Walker Asset Management Corporation (best known for Priceline.com): **36**
12. Number of patents in which the name “Amazon.com” is mentioned: **9**
13. Number of patents held by Amazon.com: **7**
14. Number of patents that mention the word “Linux”: **49**

15. Percentage of those in which Linux is used to demonstrate the patent's purpose: **100%**
16. Revenues from patent licensing in 1990: **$15 billion US**
17. Revenues from patent licensing in 1998: **$100 billion US**
18. Number of shares of Microsoft stock owned by Bill Gates: **780 million**
19. Number of shares of Microsoft stock owned by Paul Allen: **260 million**
20. Number of shares Bill Gates filed with securities regulators to sell: **300,000**
21. Percentage change in the stock price of RHAT in the five weeks after the antitrust ruling: **+218**
22. Number of feet by which Arctic sea ice has thinned since 1976: **4**
23. Number of colonies of Antarctic Adélie penguins that have disappeared since 1988: **11**

Sources

- 1-15: United States Patent & Trademark Office
- 16-17: *Upside*
- 18-20: *PC Week*
- 21-23: *Harper's Magazine*

HOWTO HAIKUS

They shine like pearls in the bathwater, these seventeen-word “found haikus” that Don Marti extracted from the Linux HOWTOs. All are syllable counts for words that appear in the CMU pronounciation dictionary.

A super daemonis bloat for those who onlywant one small feature—Werner Hauser, Linux Laptop HOWTO

I suppose you haveto fiddle around a bitto get this working—Werner Hauser, Linux Laptop HOWTO

It only takes auser with a modem tocompromise your LAN—Mark Grennan, Firewall and Proxy Server HOWTO

Examples of smoothrunning existing systemsare also welcome—Stein Gjoen, HOWTO: Multi Disk System Tuning

CD-ROMs have aspiraling track much like anaudio record—Skip Rye, Optical Disk HOWTO

Rest assured that theycan determine that it's thereand will exploit it—Kevin Fenzi & Dave Wreski, Linux Security HOWTO

The one conditionis that credit is givenwhere credit is due—Harvey J. Stein, The UPS HOWTO

-Doc Searls

THEY SAID IT

We have a highly skilled group of patent examiners with a technical background that matches up very well with the kind of technologies they are seeing—and we think we issue patents of an appropriate breadth.

—U.S. Patent & Trademark Office Commissioner Q. Todd Dickinson in *IP Worldwide*

Amazon.com Patents Enemy-Making Process

—Headline in *The Industry Standard*

Windows CE is an environment where Microsoft says, “This is what the reference design looks like, and as long as you build this, Windows CE will work on it.” Linux, on the other hand, is an erector set, where we design the reference hardware to meet the problem we're trying to solve and then go to Linux for the piece parts from the bin to build exactly what works.

—John Bork of Intel in an interview with *Linux Journal*

Open Source developers understand UNIX. This is part of what made it possible to create a better UNIX: Linux. In order to create a better MS Office, Open Source developers need to understand MS Office in as much detail as they understood UNIX. My fear is that the Open Source developer community doesn't understand Office. It can't create what it doesn't understand. What we need are more developers using Windows and Office.

—Larry Augustin, VA Linux Systems, at the New York New Media Association

SMILE, YOU'RE ON A ONE-PIXEL CAMERA!

Web pages use a publishing metaphor—they are pages, after all. We write, open, read and bookmark them. We assume when a page downloads from a server, it's a one-way deal. The HTML describes the page, lays out the print, loads the graphics onto the page and into the cache. There is the presumption of privacy. After all, this is a published page, and reading is a personal, even an intimate, act. At those times when interaction is required, such as when we fill out a form, there's a “submit” button that sends information back to the other end of the line. We're still in control.

Most of us know how cookies work. They carry the potential for evil, but most serious e-commerce sites are careful not to abuse a customer's trust. But the truth is, we are being watched—a lot—and not just by cookies. The following three web sites will be of interest if you are concerned about this issue—and you should be.

- If you aren't doing some kind of web ad blocking, please see Richard M. Smith's pages at http://www.tiac.net/users/smiths/.
- You might be especially interested in the Web Bugs FAQ at www.tiac.net/users/smiths/privacy/wbfaq.htm.
- Have your Internet privacy analyzed at http://www.privacy.net/.

It turns out that some companies are including 1x1 transparent GIFs from the web ad agencies on some pages, so you can be tracked on pages where there's no visible ad. Example: http://www.fedex.com/us/tracking/—FedEx's package tracking page. (Who's tracking who?)

Please implement some kind of banner blocking, whether it's Junkbuster, the "webclean" configuration file for Apache proxy, Squid, or just making your name server authoritative for the domain names of the big web advertising agencies.

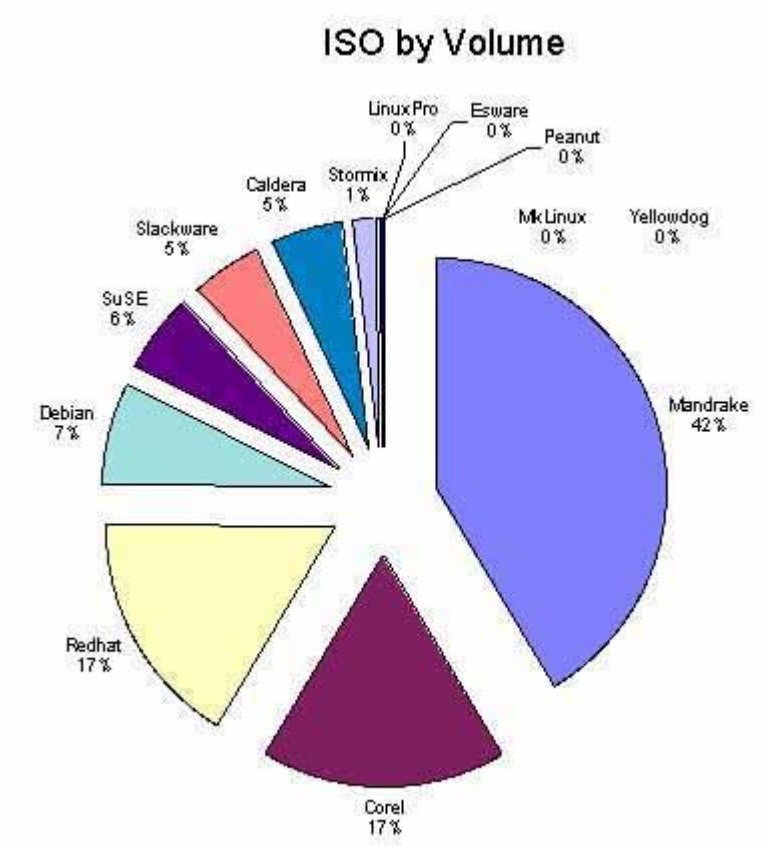
—Doc Searls

MORE SLASHQUOTES

Light travels faster than sound. That's why some people appear bright until you hear them speak:

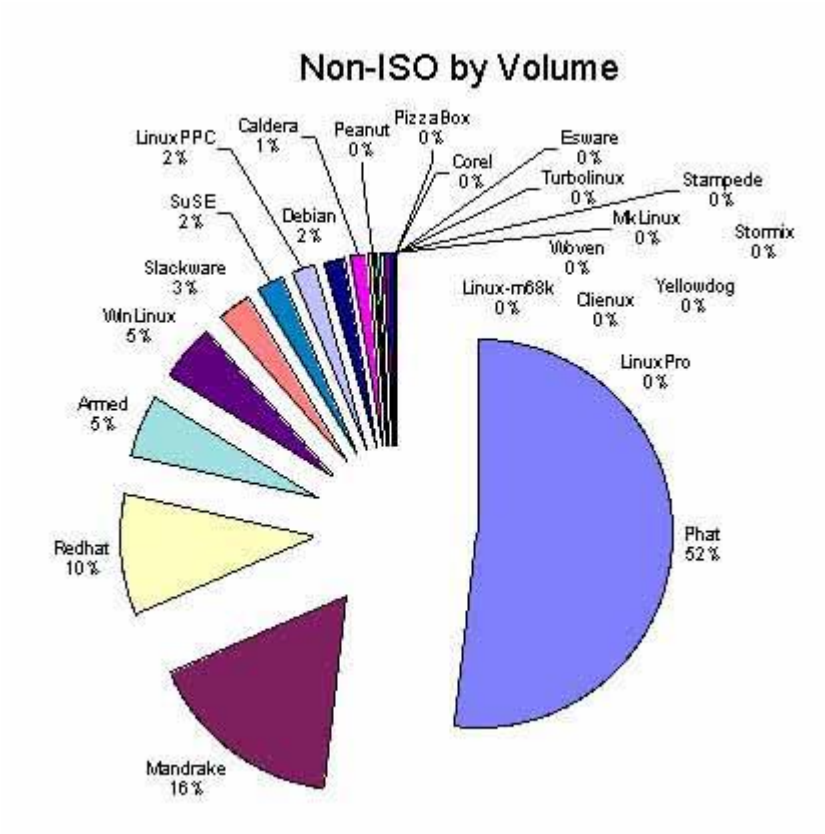
- Me fail English? That's unpossible!
- Engineers never lie; we just approximate the truth.
- this .sig no verb
- Windows 2001: "I'm sorry Dave, I can't do that."
- Eggs don't grow on trees.
- Blessed is him who, having nothing to say, abstains from giving us wordy evidence of the fact.
- Someone had to put all that chaos there!
- Linux just happens to be one of the best forking kernels out there.
- Stupidity should be painful.
- Wouldn't 1/0 approach infinity?
- militant agnostic: I don't know, and you don't know either.

TUCOWS STATISTICS

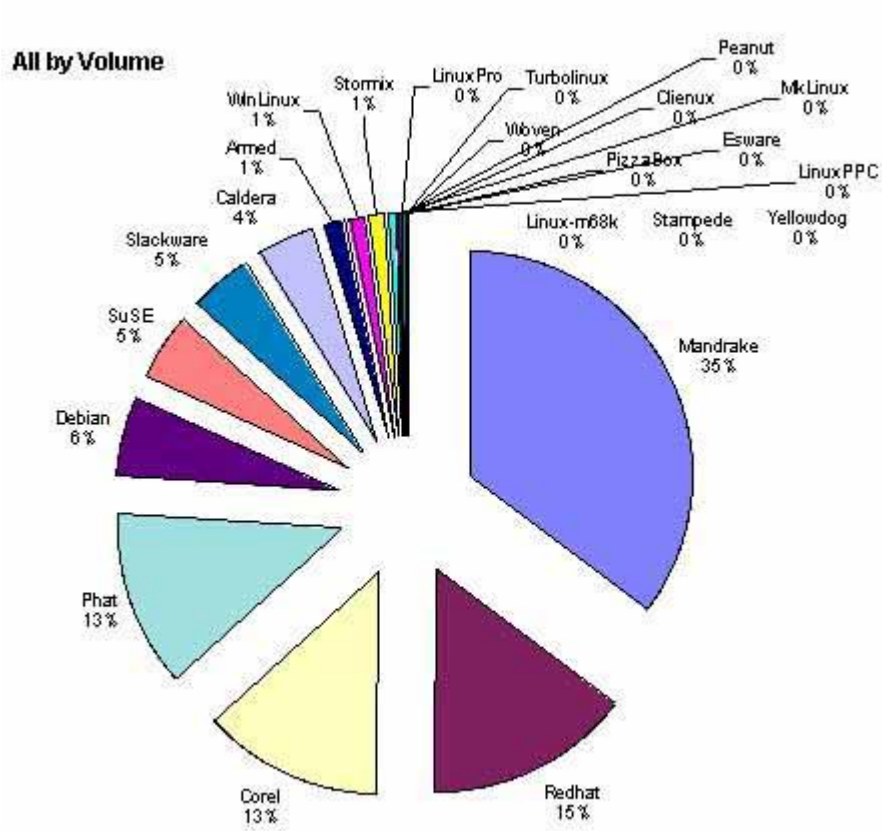


ISO downloads by Volume:

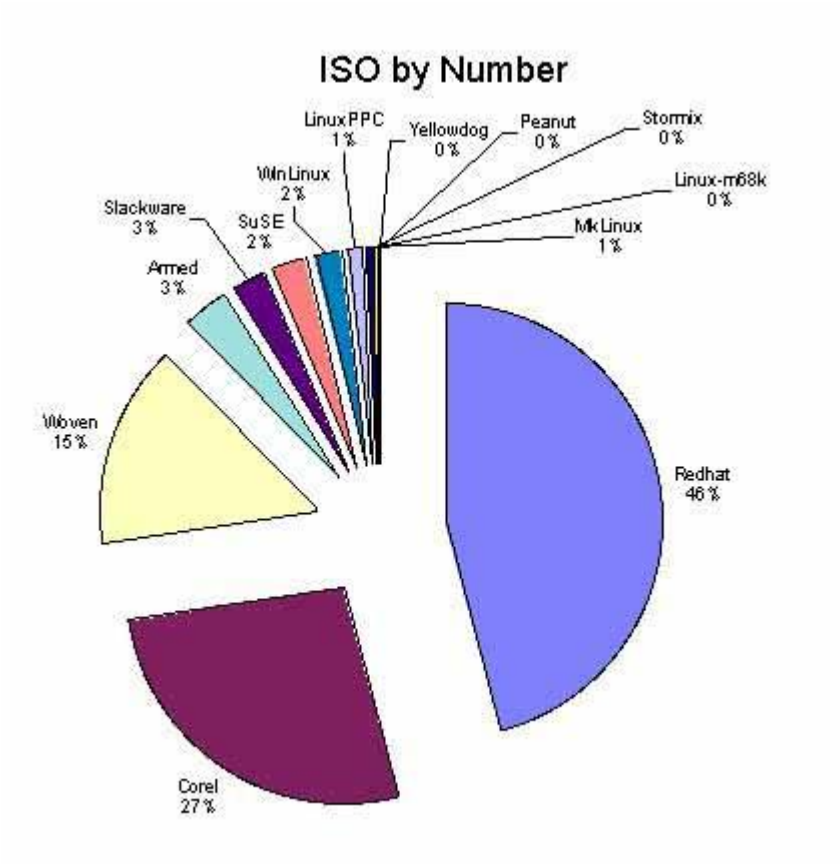
the Mandrake downflow from Tucows continues to be huge: still #1, though down 4% from January. Corel edged past Red Hat for #2, though both rounded to 17%. That's up 3% for Corel and down 1% for Red Hat. #4 Debian is up 2% and #5 SuSE is down 1%. Slackware was up 1%, Caldera held even, and Stormix showed up for the first time with a 1% wedge of the pie.



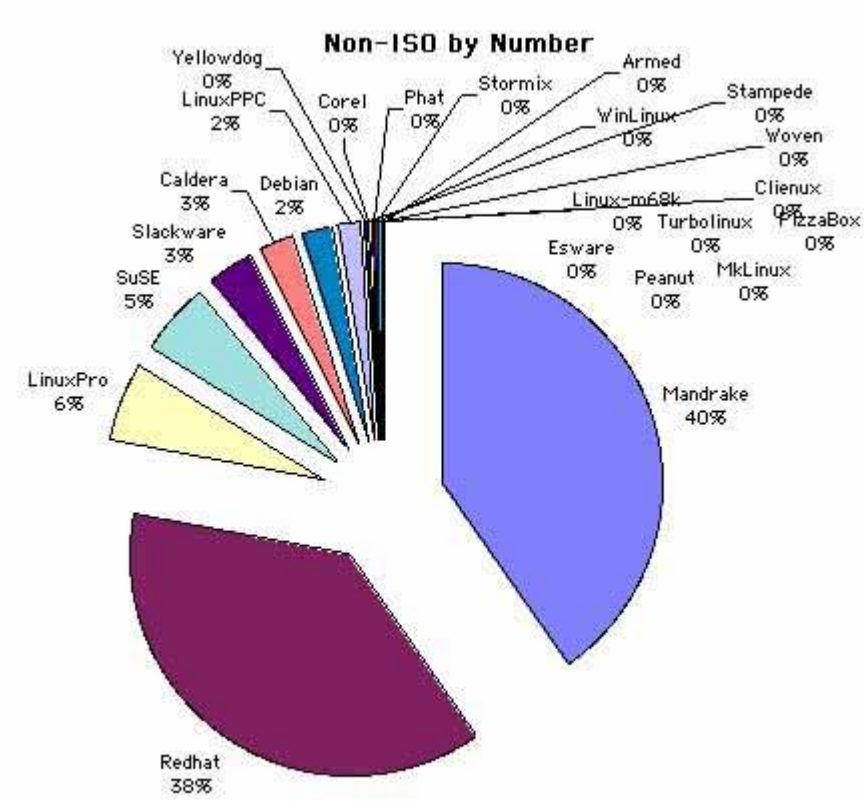
Non-ISO by Volume



All by Volume



ISO by Number



Non-ISO by Number

STRICTLY ON-LINE

A Real-Time Data Plotting Program by David Watt is an introduction to programming using the QT windowing system in X. Mr. Watt has written a real-time plotter application called RTP and tells us how he did it. This is freely available software, and you can join others in adding enhancements or use it to write your own application.

The Network Block Device by P. T. Breuer, A. Marín Lopez and Arturo García Ares tells us about this system component and how it can be used. Basically, an NBD driver will make a remote resource look as if it is a local device to Linux. Thus, it can be used to construct a cheap and safe real-time mirror.

Shell Functions and Path Variables, Part 3 by Stephen Collyer is the final article in our series to introduce you to path variables and elements. This month, Mr. Collyer talks about the **makepath** utility, more path-handling functions and a few implementation issues.

Linux Administration for Dummies is a book review by Harvey Friedman who gives us a taste of what this book is about and whether we should buy it.

WordPerfect for Linux Bible is another book review by Ben Crowder. WordPerfect is one of the most common word processors available. If you need help with this application, this book may be a good resource for you.

Python Programming for Beginners by Jacek Artymiak is a great introduction to this popular scripting language. A tutorial with many examples to help you learn the right way to code non-trial applications using Python. Once you've read it, you'll be ready to outsmart the Spanish Inquisition.

Python Conference Report is just that: a report on the conference held in Washington in January. Find out all about it in this article by Andrew M. Kuchling.

THE BUZZ

During the month of February (and the beginning of March), people were talking about:

- Microsoft's convenient web-based system on their site that allows vendors to submit invoices. Michael Olson was there and found this message: “Note: Microsoft Invoice is not compatible with Netscape Navigator, Apple MacIntosh computers, or Linux.”
- Copyleft, a company dedicated to helping further Open Source idealism, and their donation of $10,000 US to support the Electronic Frontier

Foundation (EFF). The money is intended to aid the legal defense the EFF is mounting in response to lawsuits by the Motion Picture Association of America (MPAA) and the DVD Copy Control Association (DVD CCA). Show your appreciation: buy a T-shirt. Visit their site at http://www.copyleft.net/.

- Rumors that Microsoft is considering porting Office to Linux. Arthur Tyde, executive vice president at Linuxcare, was told there are 34 developers working on this very thing at Microsoft.
- The work done by Penguin Radio, Inc. and Ineva.com work on a Linux-based car radio with the capacity to receive thousands of stations. The radio hooks up with the the Ellipso Satellite Internet service, which uses a unique (and patented) system of satellite orbits to provide global Internet connectivity. This won't happen until 2002. Huff!
- An e-mail concerning a new ad for Microsoft's Internet Explorer e-mail program. The ad uses “Confutatis Maledictis” from Mozart's Requiem as its theme music. The chorus sings “Confutatis maledictis, flammis acribus addictis.” This translates to “The damned and accursed are convicted to the flames of hell.”

STOP THE PRESSES: The Patent Conversation

> As we enjoy great advantages from the inventions of others, we should be glad of an opportunity to serve others by any invention of ours; and this we should do freely and generously. —Benjamin Franklin

Patents are mines. They lay buried in the marketplace, doing nothing until their owners blow them up under an enemy. That's what Amazon.com founder and CEO Jeff Bezos did last fall, when Barnesandnoble.com stepped too close to Amazon's “1-click” patent (No. 5,960,411). Amazon pressed a charge, and a court injunction followed.

The bomb worked. It stopped Barnesandnoble.com from copying Amazon's work on the 1-click feature. It also blew Jeff's clothes off. *Time* magazine's Man of the Year—the leading entrepreneur of the New Economy—was exposed as an old-fashioned emperor of industry: a hard-ball player, a pointy-haired litigator.

At least, that's the way it appeared to the Open Source world, where captains of industry get the benefit of little doubt in any case. Calling almost immediately for a boycott, the conscience of free software, Richard Stallman, said:

> This is an attack against the World Wide Web and against e-commerce in general. The idea in question is that a company can give you something which you can subsequently show them to identify yourself for credit. This is nothing new: a physical credit card does the

> same job, after all. But the U.S. Patent Office issues patents on obvious and well-known ideas every day. Sometimes the result is a disaster.

Not much happened after that, at least on the surface. Behind the scenes, however, Tim O'Reilly of O'Reilly & Associates began a correspondence with Jeff Bezos.

Then, on February 23, all hell broke loose. The United States Patent & Trademark Office assigned Amazon patent No. 6,029,141 for an “Internet-based customer referral system”. This covered Amazon's popular associates program, by which thousands of sites mount “bookstores” with sales fulfilled by Amazon.com. (To witness the program's popularity, search for the phrase “in association with amazon.com”.) There are many such programs operating on the Web, but Amazon's was the first and easily the most successful.

Amazon had applied for the patent back in 1997, but that didn't cut much ice with the open-source folks. A roar went up, and this time the press joined in. “Loss-making Amazon turns to bullying”, wrote the *Irish Times*. For the first time in its short life, Amazon.com was getting bad PR. If they hadn't sued Barnesandnoble.com over a different patent, it is unlikely anyone would have cared. In total, Amazon holds only seven patents. By contrast, IBM obtained 2,697 patents in 1998 alone.

Amazon's patents were all applied for prior to the land rush on “business process” and software patents after the floodgates were opened by State Street Bank & Trust Co. vs. Signature Financial Group Inc. On July 23, 1998, the Federal Court of Appeals upheld a lower court ruling that threw out what little discretion remained on the patenting of, well, nearly every marginally original way of doing business, including “processes” conducted by software. The Supreme Court later declined to review the case.

So Amazon stood alone in the spotlight. Yes, other Internet companies had sued to protect patents, but Amazon was the leading e-commerce innovator. Their patent policy mattered.

Enter Tim O'Reilly. In a series of “Ask Tim” columns and open letters, O'Reilly both challenged Bezos and invited him into the growing patent reform movement. A series of private conversations followed, culminating on March 10 with “An open letter from Jeff Bezos on the subject of patents” (http://www.amazon.com/patents/).

Crediting influence by O'Reilly, Bezos declared both his intent not to harm software development and a newfound commitment to patent reform. He did not hedge on his own patent policies, but neither did he appear to push them

off the negotiating table. He also included a number of concrete proposals for reforming patent law.

Suddenly, the PR turned around. Columnist Dan Gillmor of the *San Jose Mercury News*, who had labeled Bezos one of technology's "villains" after the lawsuit against Barnesandnoble.com, wrote, "Bezos has a point on patents..." and "...it's a heartening sign of the Internet's power, and Bezos' management style, that this conversation is taking place at all."

Will Bezos continue to straddle the fence? He faces a clear choice between his lawyers and his market. As both he and O'Reilly point out (crediting *The Cluetrain Manifesto*), markets are conversations. If he stops talking, we'll know his choice.

—Doc Searls

VENDOR NEWS

Andover.net and **T.C.X DataKonsult AB**, publisher of the MySQL relational database, announced a joint program to implement database replication in MySQL. Andover will provide financial assistance, source code and technical assistance to the MySQL team. All enhancements will be made available to all users of the product. More detailed information can be obtained at www.andover.net/ and www.mysql.com.

Linux Support Group, LLC announced plans to open its Silicon Valley Support Center in San Jose, CA. The center, modeled after LSG's Delaware-based service center, will help expand relationships to customers throughout the Western region. The support center will offer 24 x 7 technical support, technical training (LSG University) and the LSG Laboratory, which will focus on vendor-neutral product testing, certification and benchmarking.

VA Linux Systems, Inc. introduced the SourceForge CompileFarm—a service that offers open-source developers a convenient way to build and test applications on multiple versions of the Linux and BSD operating systems over the Internet. The CompileFarm will allow testing on Red Hat Linux, Debian GNU/Linux, Caldera and Slackware, as well as FreeBSD, with plans to offer SuSE and other distributions soon. SourceForge is the world's largest open-source development center, hosting over 2,500 open-source projects (http://www.sourceforge.net/).

SuSE Linux AG, Europe's leading Linux distributor, and **Qarbon.com** launched the "Linux Viewlet Project". The project will provide Linux users and developers with a free database of Viewlets addressing a wide range of Linux questions.

Viewlets, originated by Qarbon.com, are a web innovation that change help files and FAQs into demonstrations that show the user how to perform specific computing tasks. SuSE is the first Linux distribution to offer Viewlets. Qarbon.com encourages individuals to create a wide variety of Viewlets. Authors are compensated. Detailed information can be obtained at http://www.teach2earn.com/linux/.

MontaVista Software Inc., developer of the Hard Hat Linux operating system for embedded computers, announced the appointment of David Warner as chief financial officer. Mr. Warner will be key in addressing sales and distribution of the company's Hard Hat Linux subscriptions in OEM segments, including Internet appliances, communications infrastructure, industrial control and defense.

LinuxVoodoo.com is a free technical support site dedicated to providing a forum to discuss everything Linux. The site offers a searchable database of Linux HOWTO's, a 24-hour response help desk, message boards, links and more.

Gateway has committed to make a $25 million investment in **eSoft, Inc.**, which develops and markets the TEAM Internet Linux software suite. The move is Gateway's first step into the Linux arena. Gateway will provide turn-key Internet access solutions to small businesses and will leverage eSoft's network Internet service solutions to advance its commitment to small business' growing software and Internet connectivity needs. Detailed information can be obtained at www.esoft.com/ or www.gateway.com

Linuxcare, Inc., a provider of comprehensive services for Linux, announced the opening of its third European office, located in Hamburg, Germany. The new office will focus on delivering Linuxcare's services and support expertise to independent software vendors, application vendors, dot-coms and the region's largest companies deploying Linux or other open-source software. Linuxcare will localize its service offerings for German and other European clients, providing complete, vendor-independent Linux solutions to jump-start customers' e-business initiatives.

EMUmail, creators of the EMU web-mail engine, announced a program that would wave the setup fee for Linux-based domain names added to its outsourced e-mail system. EMUmail is extending its premium e-mail outsourcing program to the Linux community in an effort to help propagate Linux awareness through e-mail communication. For more information, visit their web site at http://www.emumail.com/.

Samsung Electro-Mechanics Co, Ltd. and **Lineo, Inc.**, the leading developer of embedded Linux system software, announced a partnership to use Lineo Embedix Linux as the operating system for Samsung's embedded Internet appliances. Samsung will initially use Embedix Linux and Embedix Browser in PDAs and set-top boxes. The partnership also includes joint education, sales and marketing efforts for the Korean and world-wide market.

Eazel, Inc., working with the **GNOME** development team, unveiled plans to develop products and services which will make the power and reliability of the Linux operating system accessible to mainstream desktop users. Eazel was founded in August 1999 and is led by a group of industry veterans, all of whom were part of the original Apple Macintosh team. Eazel is developing innovative desktop software for Linux which will be integrated with Internet-based services and will be released this summer.

Rumor: Microsoft insurance has been hit with a rash of psychological visits for acute orthinophobia, the fear of birds—penguins!

Quote of the Month: “We don't want to be another Netscape,” Jeff Bezos, Amazon.com in an interview with Tim O'Reilly.

Web site of the Month: Help SETI look for ET. Sign up at http://setiathome.berkely.edu/, then join the *Linux JournalReaders' group.*

Factoid: *LJ* Assistant Editor poses nude for cover of Python Supplement.

Factoid: Guido van Rossum's favorite TV show of all time: “Monty Python's Flying Circus”.

Archive Index Issue Table of Contents

Advanced search

Getcha Program!

Peter H. Salus

Issue #73, May 2000

Writing good, useful programs that aren't excessively verbose and flabby isn't easy.

Inside and outside ballparks and arenas, there are always people vending programs. Most of the computing we do is the result of other folks' programs and of programs we tweak or rewrite. That's fairly obvious.

However, as I've written before, writing good, useful programs that aren't excessively verbose and flabby isn't easy. Despite a number of really splendid books (Fred Brooks', Knuth's three volumes, Kernighan and Plauger, Kernighan and Pike, Bentley), looking at code shows a lot of untidy, repetitious slovenly work.

I admit to taking old code, snipping a few lines, typing something in, and voil<\#224>--a new driver. Fast and sloppy.

Looking at the 40 million lines of Office 2000, I realize just where the difference between open source and Windows lies—perhaps where the chasm is.

I've pointed out that if you go to the BSD manuals, you can count the number of contributors to the CSRG's (Computer Systems Research Group) efforts over 15 years. It comes to about 600 individuals. In 1998, I heard Bill Gates refer to 2000 programmers in Redmond, WA. There are most likely far more now.

Good programs are written by individuals or small groups. Brian Kernighan told me that AWK was the most difficult thing he'd ever worked on, “because there were three of us.”

Take a bunch of bright programmers, assign them to teams, assign a product to each team, and put them to work. The results will be slightly inferior to what the most feeble-minded member of each team would have produced.

I was surprised when I read *The Pragmatic Programmer* by Hunt and Thomas (Addison-Wesley, 2000). It's a nicely done volume which is intended to take the programmer from a “requirement” to a real “product”. I must admit I'd always thought of this kind of thing as somewhat comic, but it may be that Hunt and Thomas have good ideas. The book is certainly worth reading.

With all this in mind, I started thinking about the tools I use every day: shells, sed, awk, grep, sort, pipe, etc.

There are several decent books on shell programming, but none covers even most of the available shells (sh, bash, csh, ksh, tcsh and zsh come to mind, and I know there are many more). Robbins' AWK volume is excellent and there was a book on sed a few years ago, but I know of none on grep (nor egrep, which searches using regular expressions). In fact, I know of nothing on zsh. In these cases, we are actually driven to read the on-line man pages.

Just how we get to tools in everyday use is, I think, instructive. I still use **troff**, for example.

In 1967, L.P. Deutsch and B.W. Lampson took the TECO (an editor) that was running on the PDP-1 at MIT and implemented it on the SDS 940 as QED (= Quick Editor). Ken Thompson took QED and rewrote it for CTSS (compatible time sharing system) on the IBM 7094. He and Dennis Ritchie wrote a version of it for the GE 635 at Bell Labs. Thompson then took that version to create **ed** in August 1969.

That allowed for editing, but what about printing? J.E. Saltzer had written **runoff** for CTSS. Bob Morris moved it to the 635, and called it **roff**. Ritchie rewrote that as **rf** for the PDP-7, before there was UNIX (it was an evolutionary dead end). At the same time, the summer of 1969, Doug McIlroy rewrote roff in BCPL (Basic Combined Programming Language by Martin Richards, 1966), extending and simplifying it at the same time. It was McIlroy's version that first Joe Ossanna and, after his death, Brian Kernighan turned into the troff we still use.

(By the way, TECO is also the ancestor of Emacs.)

Nearly a decade later, Ted Dolotta created the memorandum (**-mm**) macros, with a lot of input from John Mashey. Thereafter, Eric Allman wrote the BSD **-me** macros. I still use them.

No teams. No large groups. Just a lot of good programmers trying to make code into something truly useful. And the open way does that.

Peter H. Salus , the author of *A Quarter Century of UNIX* and *Casting the Net*, is an *LJ* contributing editor. He can be reached at peter@usenix.org.

Archive Index Issue Table of Contents

Advanced search

Patent Absurdities

Doc Searls

Issue #73, May 2000

“We hold these truths to be self-evident, that all men are created equal, that they are endowed by their Creator with certain inalienable Rights, that among these are Life, Liberty, and the pursuit of....” —Thomas Jefferson

The last word in the first draft of Thomas Jefferson's most famous sentence was “property”, not “happiness” Perhaps that's because Jefferson always had a problem stretching the meaning of “property” over things that are not material —especially the good ideas we call inventions.

An inventor himself, Jefferson administered the country's first patent law in 1790. Historian David F. Noble says Jefferson and his colleagues applied the law “under very strict standards, and relatively few patents were issued.” Jefferson explained some of his concerns in a letter to Isaac McPherson in 1813. The letter concerned a disputed invention by Oliver Evans, called a “hopper-boy”.

After detailing the debts Mr. Evans' invention owed to past inventors ranging from Archimedes to a Virginia farmer, Jefferson wearily wrote, “The hopper-boy is a useful machine, and so far as I know, original.” Then he launched into a digression that has haunted patent law ever since, and it also described principles of creation that would not see full expression until they produced the Internet nearly two hundred years later:

> It would be curious then, if an idea, the fugitive fermentation of an individual brain, could, of natural right, be claimed in exclusive and stable property. If nature has made any one thing less susceptible than all others of exclusive property, it is the action of the thinking power called an idea.

> The moment [an idea] is divulged, it forces itself into the possession of everyone, and the receiver cannot dispossess himself of it. He who receives an idea from me, receives instruction himself without lessening

> mine; as he who lights his taper at mine, receives light without darkening me.

> That ideas should freely spread from one to another over the globe seems to have been peculiarly and benevolently designed by nature, when she made them, like fire, expansible over all space, without lessening their density at any point, and like the air in which we breathe, move, and have our physical being, incapable of confinement or exclusive appropriation.

> Inventions then cannot, in nature, be a subject of property.

Coming early in the Industrial Age, this was not a catchy idea. The truth of it may have been self-evident to Jefferson, but it was lost on nearly everyone else. The material nature of property was beyond dispute. Even Karl Marx, that great enemy of private property, was a devout materialist who made a big deal about who ought to own what.

Now Jefferson's second revolution is underway. While the first one overthrew a colonial government, the second one overthrows an industrial economy. While both sought to create ideal conditions for free expression, the second one does it by creating a whole new world—one based on the principles Jefferson described in his letter to McPherson.

By the terms of these principles, this revolution is not about e-commerce, dot-com startups or money lavished like free cocaine on every zero billion-dollar entrepreneurial impulse that connects with the venture capital syndicate on Sand Hill Road. It's about the kindlings of wealth those venture dollars ignite: *ideas* which catch and spread across the globe, illuminating everything and darkening nothing—and *inventions* which put those ideas to work.

This new world, the Internet, is a work of nature—*our* nature, as social and original beings. It is, without a doubt, the best thing we have ever done for ourselves. It connects across every space that divides us and gives us efficient new ways to meet, talk, teach, learn, gossip, argue, tell stories and do business.

As a natural habitat for business, the Internet is utopia. For proof, just step back twenty years and take a look. The threshold of enterprise is approximately zero. New business concepts catch and spread at the speed of gossip. The average distance from idea to wealth is reduced from decades to months, millionaires are minted by the minute and billionaires by the day. Vast quantities of fresh money are lavished on every credible ambition, with instructions to set bonfires with it in the midst of the marketplace, without collateral and with minimal personal risk. New companies with negligible revenues are valued entirely for their promise without regard for material

intrinsics, turning Wall Street into a casino where the house hardly stands a chance (and in fact does not exist).

In the midst of it all, unprecedented sums of work get done.

The great irony here is that this utopia was not built like an empire, or by people who were, in Walt Whitman's words, “consumed with the mania of owning things”. It was built like an Amish barn by hackers who made it because they needed it, and it sure wasn't going to come from the old software industry. The result was a second world—one made with code rather than matter—that embodied and expressed the long-overlooked virtues of the first:

- No one owns it.
- Everyone can use it.
- Anyone can improve it.

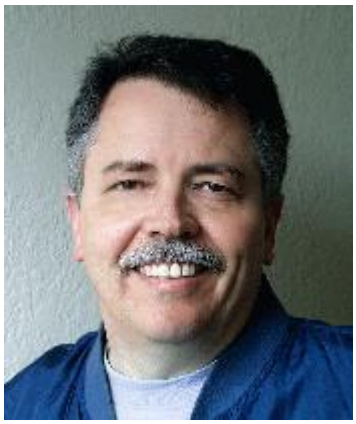
These principles are so basic, they undermine all efforts to deny them.

The gears of the old patent system can't get a purchase on this new world, even though the Supreme Court decided in 1998 that software and “business methods” were patentable. It has been amazing to watch the patent stampede that followed this wacky decision. Thousands of patents were filed to stake out claims in empty space. Now that the first of these patents are getting approved, we're starting to see lawsuits. The Amazon and Priceline suits are only the most familiar ones. There will be many more.

There will also be much garment-rending and teeth-gnashing over the “threats” these patents and lawsuits pose to our new “World of Code”. These new patents are patently outrageous, but they are also futile. Nature will take care of business.

Markets are conversations. You have to talk with them. If you try to command and control them the old-fashioned way, they'll get bored and move on. The rules of Darwin still apply: if you want to evolve, you have to adapt or else you die.

Companies that choose to evolve will need to adapt to the gift culture that produced our utopia. “Gift cultures,” Eric Raymond says, “are adaptations not to scarcity but to abundance.... In gift cultures, social status is determined not by what you control but by *what you give away*.” The italics are his. This gift is ours.



Doc Searls (info@linuxjournal.com) is Senior Editor of *Linux Journal* and co-author of *The Cluetrain Manifesto*.

Archive Index Issue Table of Contents

Advanced search

Best of Technical Support

Various

Issue #73, May 2000

Our experts answer your technical questions.

Who Goes There?

I have taken over a Linux system for a company. The company uses the box as an Internet server. The problem is that the person I have taken over for left without providing the root password for the box. Is there any way I can find the password, or any way I can recreate it without knowing the original password? —Rusty Mays, rmays@ntr.net

When you get the LILO prompt, press the tab key. Find which kernel name is first (probably “linux”) and type

```
linux init=/bin/bash
```

Once you get a prompt, do the following:

```
mount -wno remount /
/bin/vi /etc/passwd
```

The old trick was to remove the password field (anything between the 1st and the 2nd colon), but some distributions (notably Red Hat) have a bug (or feature, depending on how you look at it) where it won't let you log in as root if there is no password, and also the password command will not work if the password field is blank. The **passwd** command should work and let you enter a new password even if you don't know the old one (and leave it in place). If that fails, another option is to replace the encrypted password with a known encrypted one, like yours. —Marc Merlin, marc_bts@valinux.com

The passwords used in UNIX/Linux are usually hard to crack, so the best thing to do is to boot from a Linux boot floppy, then edit the password file. Boot from a set of Linux boot floppies, either for an installation or one of the “Linux on a

floppy" installations from http://www.toms.net/rb/. After booting from the floppy, mount the hard disc, then make a copy of the password file, just in case (either /etc/passwd or /etc/shadow), then edit the password file (with vi, pico, etc., or use sed and/or cut) to remove the password for root. Finally, unmount the hard disk and reboot, then log in as root and set the root password immediately. It's probably a good idea to check for other accounts with a UID of 0, and accounts you do not recognize. —Keith Trollope, kt57707@GlaxoWellcome.co.uk

I Can't Get No Resolution

My machine at home has a sis6215 chip, and Red Hat 6.0 doesn't have a driver for it. Probably because of this, when Gnome runs, the screen is zoomed and everything is huge. The default resolution is some 320x200. I tried to configure custom settings through Xconfigurator and also X11Config, but the resolution didn't change. Where am I stuck? Is there some problem with my monitor settings? —Pankaj Ratan Lal, prlal@quantum.ac.in

According to www.xfree86.org/3.3.6/SiS2.html#2, your chip isn't supported by XFree86. You can still try XFree86 3.3.6, but there aren't many chances it will work. If your chip is VESA 2-compliant, you have another option: use the VesaFB frame buffer, available at www.xfree86.org/FAQ/#FBDev. --Marc Merlin, marc_bts@valinux.com

The Dreaded Busy Signal

When I try to set up my modem and use the query modem function, I get a "modem busy" response. This also happens if I use dialup. I use COM3 for my modem, so I set Linux to tty2. I think this is correct. Is it conflicting with Windows? Any suggestions? —Don Hoornaert, richey@mb.sympatico.ca

Even though many things could be happening (misconfigured port settings, broken modem, misconfigured software, etc.), make sure you have all settings correct and try connecting directly to the modem with **cu -l /dev/tty*ofyourmodem***. The "**ttyofyourmodem**" depends on which COM port you use; it could be ttyS0, ttyS1, ttyS2 or ttyS3 for COM1, COM2, COM3 or COM4, respectively. If you succeed in connecting to the modem this way and get an OK in response to an **AT** command, then you are okay. If you bought your computer with Windows pre-installed, it is probable that you have what is called a Winmodem, which is a crippled kind of modem for Windows only. If that is the case, you had better buy a modem that works with Linux. I have bought many 3COM USRobotics "Python" internal modem cards that provide 56K V.90 data/fax/voice services to my Linux servers with excellent results. Check this excellent web page for modem compatibilities: www.o2.net/~gromitkc/winmodem.html. Check the entire modem list table. There are some

people trying to make Winmodems work with Linux, but to play safe, buy a new Linux-compatible one. —Felipe E. Barousse, fbarousse@piensa.com

Two Many Drives

I have two CD-ROM drives, one recently installed. Before the installation of the second CD-ROM which is an HP 9200i CD_RW, everything mounted okay. Now, when I issue the **mount** command (**mount /mnt/cdrom**), I get the following:

```
CD-ROM I/O error: dev 0b:00,sector 64 isofs_read-super: bread
failed,dev 0b:00 iso_blknum 16 block 32 mount: wrong fs type,
bad option, bad superblock on /dev/cdrom or too many file systems
```

Both are SCSI drives. Please help if you can. —Tom Mcloughlin, tommcl@pacbell.net

There are two likely reasons for this problem. Either your “second” CD-ROM is seen as “first”, or the devices are configured to use the same SCSI ID, thus preventing any of them from functioning. Please check how SCSI identifiers have been assigned to the drivers and remember that the smaller ID is considered “first” drive (/dev/sr0) and the bigger ID is considered “second” (/dev/sr1). —Alessandro Rubini, rubini@linux.it

/dev/cdrom is a symbolic link to a device like /dev/scd0 (the first SCSI CD-ROM). The second SCSI CD-ROM will be /dev/scd1. So, when wanting to mount the second CD-ROM, issue the command **mount /dev/scd1 /mnt/cdrom**, or create new links like /dev/cdrom0 to /dev/scd0 and /dev/cdrom1 to /dev/scd1. Don't forget to update the /etc/fstab, because mount **/mnt/cdrom** will read the fstab and mount the device allocated to this mounting point. —Paulo J V Wollny, paulo@wollny.com.br

Cannot Print from Linux

I have the following operating systems installed on my PII Intel 440Bx system: NT4/Windows 98/Linux. While I can print absolutely fine from Windows 98 and NT4 on my HP 670C printer, I cannot do so from Linux. The error I get is “printer port is not recognised”. I have tried using the **printtool** to force the HP 670C on lp0, lp1 and lp2, respectively, but to no avail. What could be the problem? I have even tried changing the BIOS setting between ECP and EPP. —Sunil Dhaka, dax@net4india.com

Are your printers parallel? If so, you might not have the parallel printer port driver configured correctly. Sometimes with Red Hat, the parallel port is not set up correctly after the installation. Add this line to the /etc/conf.modules file:

```
alias parport_lowlevel parport_pc
```

Reinitialize your machine, and your parallel port should now work fine. Configure everything with the printtool utility and send a test page to print. —Felipe E. Barousse, fbarousse@piensa.com

A local user I know, Scott Hettel, recently solved this problem on his own system. He found that Red Hat 6.1 does not support the IBM PC parallel printer port by default. The answer for this is on Red Hat's web site. See bug numbers 5698 and 5821 on their site for the resolution to this problem. You may also want to look at bug 8969, which discusses port compatibility. Please note that these issues affect only version 6.1 of Red Hat. —Chad Robinson, Chad.Robinson@brt.com

Swimming Up River

I'm trying to use **mgetty** with a serial port attached to a modem. The port is /dev/ttyS1. The line I put in /etc/inittab is:

```
S1:2345:respawn:/sbin/mgetty ttyS1
```

Shortly after the system came up after reboot, I received this message continuously at about five-minute intervals:

```
INIT: Id "S1" respawning too fast: disabled for 5 minutes.
```

Any ideas as to what I'm doing wrong? I'm running Red Hat 6.0. —Chris Yeats, CYeats@limbach.com

The message you get means that Linux starts the mgetty process and for some reason it dies, gets restarted and dies again. The most probable cause is that your modem cable does not have the correct pin wiring to work. Check www.linuxdoc.org/HOWTO/Text-Terminal-HOWTO-17.html#fast_respawn to find out a bit more about the problem. To override the cable problem (but losing modem control features), use the **-r** option for mgetty, which makes mgetty not monitor and detect the missing pin signals on your cable. I would still buy a good modem cable, though. —Felipe E. Barousse, fbarousse@piensa.com

Your mgetty ttyS1 command exits immediately and init disables its further invocation. You should first make the command work from the command line, where you can check what its error messages are and whether it works as expected. You should put the command in inittab only after your problems (misconfiguration, I suppose) are fixed. —Alessandro Rubini, rubini@linux.it

Sound Support

I use a Compaq Presario which came with an ESS sound chip, but I would like to use SB Vibra 128 instead. The problem is I cannot disable the ESS chip because there aren't any jumpers. Also, it's not possible to disable from BIOS menu. Is there any way to make my new sound card work under Linux? It seems as if both sound cards want to use the same resources. I am using Slackware version 7.0. —Chumpon Thamwiwat, thchum@rocketmail.com

Having no way to change settings of the ESS sound chip, maybe you can change the settings on the Vibra hardware. The driver configuration parameters may be of help also; many times, the driver indicates to the hardware which IRQ and address the device should use. If you do not load the driver for the ESS, then the ESS won't be activated, therefore you could load the driver for the Vibra, assuming there is no conflict of hardware settings and each board has its own drivers. —Felipe E. Barousse, fbarousse@piensa.com

Where in Carmen Sandiego Is xhfs?

I used to run SuSE Linux on a meek Gateway laptop with no problem; **xhfs** installed with it. Now I have a heated ASL Labs laptop and a tweaked version of Red Hat 6.1, and *no* hfs utilities or xhfs! I searched the Web and found hfs utilities, but not XHFS. Freshmeat and RPM had never heard of it. Also, I need help making my brand-new Ethernet cable connect my Power Mac 7600 and my laptop. I see the plugs, but have no clue what to do next. I have installed NetaTalk. —Hal, hsundt3@uswest.net

The hfs utilities package containing XHFS can be found at http://www.mars.org/home/rob/proj/hfs/. As for your Ethernet cable, you'll have to start with getting TCP/IP running. Configure your Mac and your laptop to be on a common subnet. Use 192.168.100.0 netmask 255.255.255.0, for example. —Marc Merlin, marc_bts@valinux.com

Resources

Many on-line help resources are available on the SSC web pages. Sunsite mirror sites, FAQs and HOWTOs can all be found at http://www.linuxjournal.com/.

Archive Index Issue Table of Contents

Advanced search

New Products

Ellen M. Dahl

Issue #73, May 2000

IQ2 NetCONNECT, OpenLinux eServer 2.3, CommuniGate Pro, Super-Symmetric Dynamic Cluster Version and more.

IQ2 NetCONNECT

BASCOM Global Internet Services, Inc. unveiled the IQ2 NetCONNECT, which provides the needed “gateway” to connect a network to the Internet, while offering built-in, transparent web caching and an integrated firewall. It is designed to update itself automatically at night with seamless integration as a priority. BASCOM's IQ2 solutions leverage Caldera's OpenLinux OS to provide a virtually self-maintaining communications platform.

Contact: BASCOM Global Internet Services, Inc., 275-R Marcus Blvd., Hauppauge, NY 11788, 631-434-6600, 631-434-7800 (fax), sales@bascom.com, http://www.bascom.com/.

OpenLinux eServer 2.3

Caldera Systems, Inc. announced its eBusiness server platform, OpenLinux eServer 2.3. OpenLinux eServer 2.3 provides everything needed to deploy and manage Linux-based eBusiness solutions. It comes with over 20 server-specific Internet applications, ten major server products, easy web-based remote management and is pre-tuned for server-specific hardware to save configuration and administration time.

Contact: Caldera Systems, Inc., 888-465-4689, info@calderasystems.com, http://www.calderasystems.com/eserver/.

CommuniGate Pro, Super-Symmetric Dynamic Cluster Version

Stalker Software announced the Super-Symmetric Dynamic Cluster version of CommuniGate Pro. The high-end CommuniGate Pro messaging system, available for UNIX, Linux and NT platforms, can serve 200,000 accounts on single-server installations and support several million accounts on heterogeneous multi-server clusters. Key features include highly scalable cluster support (full redundancy, load balancing) multi-domain support; LDAP and ACAP support.

Contact: Stalker Software, Inc., 655 Redwood Highway, Suite 275, Mill Valley, CA 94941, 800-262-4722, 415-383-7461 (fax), sales@stalker.com, http://www.stalker.com/.

Embedix Linux 1.0

Lineo, Inc. shipped its first packaged distribution of Lineo Embedix Linux 1.0 for x86 and PowerPC. Embedix Linux allows OEM developers to test and implement embedded Linux on their specific hardware solutions. This version

of Embedix Linux includes more than 25 powerful Linux service components. Embedix Linux, based on the 2.2 kernel, uses a minimum of 8MB RAM and 3MB of ROM or Flash memory. Embedix Linux is available as a free, not-for-resale download from Lineo's web site, and on CD-ROM from Lineo's web store. Contact: Lineo, Inc., 383 South 520 West, Lindon, UT 84042, 801-426-5001, 801-426-6166 (fax), info@lineo.com, http://www.lineo.com/.

white dwarf linux 1.0

EMJ Embedded Systems introduced white dwarf linux (wd linux) 1.0. Containing over 40 software packages including a web browser, an Apache 1.3.11 web server and e-mail capability, white dwarf 1.0 is based on the most recent Linux kernel, 2.2.14. wd linux is small enough to load on embedded PCs, but dense enough to contain the features embedded applications demand. Contact: EMJ Embedded Systems, 1434 Farrington Road, Suite 100, Apex, NC 27502, 800-436-5872, 919-363-4425 (fax), emjembedded@emj.com, http://www.emjembedded.com/linux/.

Linux-based Internal G.lite Modem

Silicon Automation Systems Limited extended its Synapse range of xDSL products by introducing a G.lite solution for Linux. The Synpase G.lite for Linux modem provides data transfer rates of 1.5MB/s downstream and 512KB/s upstream while allowing simultaneous access for analog voice telephony. It is fully compliant with the ITU-T G.992.2 standard and has been inter-operated with most major DSLAM vendors.

Contact: SAS, 3008 12th B Main, HAL 2nd Stage, Bangalore 560 008, India, info@sasi.com, http://www.sasi.com/.

IntraLinux

STS International launched its new Linux product, IntraLinux, which bundles special implementation and support for small and medium-sized corporations and enterprise workgroups. IntraLinux consists of a networking solution that

includes an optimized version of the Linux OS and a number of implementation and management tools specifically designed for corporate intranets and LANs.

Contact: STS International, Inc., 321 Hartz Avenue, Suite 200, Danville, CA 94526-3336, 888-422-5787, 925-831-2810 (fax), sales@intralinux.com, http://www.intralinux.com/.

Linux End-to-End Security Solution

JAWS Technologies Inc. announced a commercially available end-to-end enterprise security solution for Linux. The new Linux-based products and services offer secure remote data storage capabilities and JAWS' proprietary data encryption products including XMail, L5 Encryption for the Desktop and an upcoming gateway server-based solution for encrypting corporate e-mail.

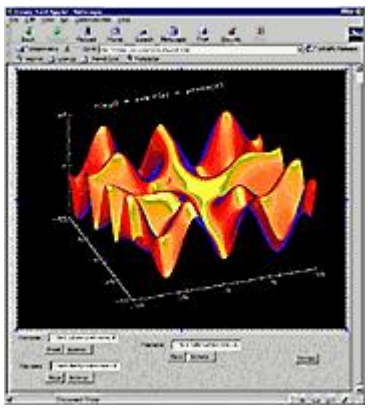
Contact: JAWS Technologies, 1013-17th Avenue SW, Calgary, Alberta T2T 0A7, Canada, 888-301-5297, 403-508-5058 (fax), info@jawstech.com, http://www.jawstech.com/.

JetForm Central for Linux

JetForm Corporation introduced JetForm Central for Linux, an electronic document output solution for producing e-business documents from existing line-of-business applications such as ERP, financial services and government systems. Linux users have control over document data and access to alternative output capabilities including web, e-mail, fax, PDF and print. Key features include distributed output capabilities, dynamic data-driven document generation, graphical design and powerful, flexible output.

Contact: JetForm Corporation, 560 Rochester Street, Ottawa, ON K1S 5K2, Canada, 800-538-3676, info@jetform.com, http://www.jetform.com/.

JWAVE version 3.0



Visual Numerics announced JWAVE 3.0, a client/server solution that uses Sun Microsystems' Java components to rapidly develop and deploy applications across an enterprise via the Internet or an intranet. These applications are 100% pure Java and let users perform numerical analysis and visual interpretation of large, complex datasets. JWAVE 3.0 includes 76 new functions from the IMSL C Numerical Library, and a comprehensive set of more than 300 pre-built mathematical and statistical analysis functions written in C which can be embedded directly into data analysis applications.

Contact: Visual Numerics, Inc., 5775 Flatiron Parkway, Suite 220, Boulder, CO 80301, 303-939-8797, 303-245-5300 (fax), http://www.vni.com/.

RTEL

Lantronix announced its free RTEL software utility with support for Linux. The Lantronix RTEL application creates a virtual device in the Linux device directory, allowing the Linux server to believe it is printing to a local printer. RTEL transparently passes print jobs over the network to any printer connected by a Lantronix print server, with all information intact. It can be combined with Samba to allow Linux servers to seamlessly spool and manage print jobs from Microsoft Windows clients.

Contact: Lantronix, 15353 Barranca Parkway, Irvine, CA 92618, 800-422-7055, 949-450-7232 (fax), sales@lantronix.com, http://www.lantronix.com/products/utils/rtel/.

Japanese WordMage v5.7

Lava Software began shipping Japanese WordMage v5.7, a complete low-cost Japanese study aid/application suite for Linux and others operating systems. Many features also support the extended European, Cyrillic and Greek character sets. It offers nine highly integrated applications including a

multilingual word processor, an HTML web page editor/viewer, various study systems with authoring abilities, a powerful Kanji reference dictionary, a grammar library builder and a text translation aid.

Contact: Lava Software Pty. Ltd., GPO Box 215, Adelaide 5001, Australia, +61-8-8235-0003, +61-8-8235-0668 (fax), service@lavasoft.com, http://www.lavasoft.com/.

TotalView 4.0 Parallel Debugger

Etnus began shipping TotalView 4.0, a parallel debugger which supports multiple development platforms for both UNIX and Linux. The GUI-based, single- and multi-process debugger shortens development time via an easy-to-learn and easy-to-use "select-and-dive" approach. TotalView enables developers to unravel and control multiple threads and processes running on single or multiple processor systems. The debugger automatically acquires related processes and threads and graphically displays data arrays.

Contact: Etnus, 111 Speen Street, Framingham, MA 01701-2090, 508-875-3030, 508-875-1517 (fax), info@etnus.com, http://www.etnus.com/.

iNUX Consumer Linux Personal Computer Systems

iNUX Inc. announced the release of a consumer and small business desktop computer, utilizing the Linux OS. iNUX Consumer Linux Personal Computer (PC) systems are based on SuSE and include iNUX's graphical user interface, designed to work with KDE, to deliver easy and intuitive access to a broad selection of pre-loaded and pre-configured applications and content.

Contact: iNUX Inc., Two Adalia Ave., Suite 706, Tampa, FL 33606-3335, 877-357-4689 (toll-free), 813-962-5900 (fax), info@inux.com, http://www.inux.com/.

LINUX JOURNAL

Dynamic Class Loading for C++ on Linux

James Norton

Issue #73, May 2000

A technique for developers that will provide them with much flexibility in design.

Linux has much to offer as a development platform: a robust operating environment with tested tools. Linux also boasts implementations of just about every programming language available. I think it is safe to say, however, that among compiled languages, C is the language of choice for most Linux developers. Consequently, other languages such as C++ seem to be somewhat neglected in most discussions of Linux development.

The dynamic class loading technique provides developers with a great deal of flexibility in their designs. Dynamic class loading is a means of providing extensibility without sacrificing robustness.

In this article, we will design a simple application that defines a single class, a shape class we wish to use in a drawing package. As we shall see, dynamic class loading allows us to provide a smooth extension path through which users of the application can add new types of shapes without needing to modify the original application code.

Polymorphism

The basic idea behind dynamic class loading is the concept of polymorphism. Anyone familiar with C++ should be familiar with this concept, so I will discuss it here only briefly. In short, polymorphism is the ability of an object belonging to a derived class to act as an object belonging to the base class. This is the familiar “is a” relationship of OOP (object-oriented programming) parlance. For example, in the following code snippet, **circle** is a class derived from the base class **shape** (see Listing 1), so the object **my_circle** can act as a shape object, invoking the shape member function **draw**.

Listing 1

```
class shape { public:
   void draw();
};
class circle : public shape { };
int main(int argc, char **argv){
   circle my_circle;
   my_circle.draw();
}
```

While this has all the usual advantages, e.g., code reuse, the real power of polymorphism comes into play when draw is declared to be virtual or pure virtual, as follows:

```
class shape{ public:
   virtual void draw()=0;
};
class circle : public shape { public:
   void draw();
}
```

Here, circle has declared its own draw function, which can define behavior appropriate for a circle. Similarly, we could define other classes derived from shape, which provide their own versions of draw. Now, because all the classes implement the shape interface, we can create collections of objects that can provide different behavior invoked in a consistent manner (calling the draw member function). An example of this is shown here.

```
shape *shape_list[3];   // the array that will
                            // pointer to our shape objects
shape[0] = new circle;  // three types of shapes
shape[1] = new square;  // we have defined
shape[2] = new triangle;
for(int i = 0; i < 3; i++){
   shape_list[i].draw();
}
```

When we invoke the draw function for each object on the list, we do not need to know anything about each object; C++ handles the details of invoking the correct version of draw. This is a very powerful technique, allowing us to provide extensibility in our designs. Now we can add new classes derived from shape to provide whatever behavior we desire. The key here is that we have separated the interface (the prototype for shape) from the implementation.

While this technique is very powerful, it does suffer from a drawback in that we must recompile (or at least relink) our code when we add new derived classes. It would be more convenient if we could simply load in new classes at runtime. Then, anyone using our code libraries could provide new shape classes (with new draw functions) without even needing our original source code. The good news, and the subject of this article, is that we can.

dlopen and Dynamic Class Loading

While C++ has no direct mechanism under Linux for loading in classes at runtime, there is a direct mechanism for loading C libraries at runtime: the dl functions **dlopen**, **dlsym**, **dlerror** and **dlclose**. These functions provide access to the dynamic linker **ld**. A complete description of these functions is provided in the appropriate man page, so they are presented here only briefly.

The prototypes for the functions are as follows:

```
void *dlopen(const char
void *dlsym(void *handle, char *symbol);
const char *dlerror();
int dlclose(void *handle);
```

The dlopen function opens the file given in *filename* so that the symbols in the file can be accessed via the dlsym function. **flag** can take one of two values: **RTLD_LAZY** or **RTLD_NOW**. If **flag** is set to **RTLD_LAZY**, dlopen returns without attempting to resolve any symbols. If **flag** is set to **RTLD_NOW**, dlopen attempts to resolve any undefined symbols in the file. Failure to resolve a symbol causes the call to fail, returning a NULL. **dlerror** can be used to provide an error message explaining the failure. The dlsym function is used to obtain a pointer to the functions (or other symbols) provided by the library. **handle** is the pointer to the thing being referenced, and **symbol** is the actual string name of the item referenced, as it is stored in the file.

Given that we can use these functions to access functions in a C library, how do we use them to access classes in a C++ library? There are several problems to overcome. One is that we must be able to locate the symbols we need in the library. This is trickier than it might seem because of the difference between the way symbols are stored in C and C++ files. Secondly, how can we create objects belonging to the classes we load? Finally, how can we access those objects in a useful manner? I will answer these three questions in reverse.

Since we do not have the prototypes for the classes we load dynamically, how can we access them in our code? The answer to this lies in the description of polymorphism in the preceding section. We access the functionality of the new classes through the common interface provided by their base class. Following the examples above, any new shape classes would provide a draw function that would allow an object of that class to render itself.

Fine; we can use pointers to the base class to access objects from the derived classes. How do we create the objects in the first place? We don't know anything about the classes that might be loaded, other than the fact that they conform to the shape interface. For instance, suppose we dynamically load a library that provides a class called **hexapod**. We can't write

```
shape *my_shape = new hexapod;
```

if we don't know the class name ahead of time.

The solution is that our main program doesn't create the objects, at least not directly. The same library that provides the class derived from shape must provide a way to create objects of the **new** class. This could be done using a **factory** class, as in the factory design pattern (see Resources) or more directly using a single function. To keep things simple, we will use a single function here. The prototype for this function is the same for all shape types:

```
shape *maker();
```

maker takes no arguments and returns a pointer to the constructed object. For our hexapod class, maker might look like this:

```
shape *maker(){
   return new hexapod;
}
```

It is perfectly legal for us to use **new** to create the object, since maker is defined in the same file as hexapod.

Now, when we use dlopen to load a library, we can use dlsym to obtain a pointer to the maker function for that class. We can then use this pointer to construct objects of the class. For example, suppose we want to dynamically link a library called libnewshapes.so which provides the hexapod class. We proceed as follows:

```
void *hndl = dlopen("libnewshapes.so", RTLD_NOW);
if(hndl == NULL){
   cerr << dlerror() << endl;
   exit(-1);
}
void *mkr = dlsym(hndl, "maker");
```

The pointer to maker must be of type **void ***, since that is the type returned by dlsym. Now we can create objects of the hexapod class by invoking **mkr**:

```
shape *my_shape = static_cast<shape *()>(mkr)();
```

We are required to cast mkr to a pointer to a function returning **shape *** when we invoke it.

Some readers may see a problem with the code as written thus far: the dlsym call is likely to fail because it cannot resolve **"maker"**. The problem is that C++ function names are mangled to support function overloading, so the maker function may have a different name in the library. We could figure out the mangling scheme and search for the mangled symbol instead, but fortunately

there is a much simpler solution. We need only tell the compiler to use C-style linkage using the **extern "C"** qualifier, as shown in Listing 2.

Listing 2

Autoregistration

Loading the maker functions into an array associates a position in the array with each maker. While this may be useful in some cases, we can obtain more flexibility using an associative array to hold the makers. The Standard Template Library (STL) **map** class works well for this, as we can then assign key values to the makers and access them via these values. For example, we may desire to assign string names to each class and use these names to invoke the appropriate maker. In this case, we can create a map such as this:

```
typedef shape *maker_ptr();
 map <string, maker_ptr> factory;
```

Now when we want to create a particular shape, we can invoke the proper maker using the shape name:

```
shape *my_shape = factory[
```

We can extend this technique to make it even more flexible. Rather than loading the class makers in and explicitly assigning a key value to them, why not let the class designers do the work for us? Using a little bit of ingenuity, we can have the makers register themselves with the factory automatically, using whatever key value the class designer chooses. (There are a couple of warnings here. The key must be of the same type as all the other keys, and the key value must be unique.)

One way to accomplish this would be to include a function in each shape library that registers the maker for us, and then call this function every time we open a shape library. (According to the dlopen man page, if your library exports a function called **_init**, this function will be executed when the library is opened. This may seem to be the ideal place to register our maker, but currently the mechanism is broken on Linux systems. The problem is a conflict with a standard linker object file, crt.o, which exports a function called _init.) As long as we are consistent with the name of this function, the mechanism works well. I prefer to forego that approach in favor of one that will register the maker simply by opening the library. This approach is known as “self-registering objects” and was introduced by Jim Beveridge (see Resources).

We can create a proxy class used solely to register our maker. The registration occurs in the constructor for the class, so we need to create only one instance

of the proxy class to register the maker. The prototype for the class is as follows:

```
class proxy {
public:
   proxy(){
      factory["shape name"] = maker;
   }
};
```

Here, we assume factory is a global map exported by the main program. Using **gcc/egcs**, we would be required to link with the **rdynamic** option to force the main program to export its symbols to the libraries loaded with dlopen.

Next, we declare one instance of the proxy:

```
proxy p;
```

Now when we open the library, we pass the **RTLD_NOW** flag to dlopen, causing **p** to be instantiated, thus registering our maker. If we want to create a circle, we invoke the circle maker like so:

```
shape *my_circle = factory["circle"];
```

The autoregistration process is powerful because it allows us to design the main program without having explicit knowledge of the classes we will support. For instance, after the main program dynamically loads any shape libraries, it could create a shape selection menu using all the keys registered in the factory. Now the user can select “circle” from a menu list, and the program will associate that selection with the proper maker. The main program does not need any information about the circle class as long as the class supports the shape API and its maker is properly defined.

Listing 3

Listings 1 through 5 pull together the concepts presented thus far. The shape class defined in Listing 1 is the base class for all shapes. Listings 2 and 3 are the source code for dynamically loadable libraries that provide circle and square shapes, respectively.

Listing 4

Listing 4 is the main program that is extensible through dynamically loaded libraries. It scans the current directory for any .so files (libraries) and opens them. The libraries then register their makers with the global factory provided by the main program. The program then dynamically constructs a menu for the user with the shape names registered by the libraries. Using the menu, the user

can construct shapes, draw the shapes constructed, or exit the program. Listing 5 is the makefile used to build the project.

Listing 5

Real-World Examples

Recently, I have had two occasions to use this technique. In the first case, I was developing a moving object simulation. I wanted to provide users with the ability to add new types of moving objects without having access to the main source. In order to accomplish this, I defined a base class called entity, which provides the interface definition for any moving object in the simulation. A simplified version of the entity prototype is shown below.

```
class entity {
private:
   float xyz[3];  // position of the object
public:
   activate(float)=0; // tell the object to move
   render()=0;  // tell the object to draw itself
};
```

Thus, all entities have at least a position in three-space, and all entities can draw themselves. Most entities will have many other state variables besides position and may have many other member functions besides **activate** and **render**, but these are not accessible through the entity interface.

New entity types can be defined, incorporating whatever motion dynamics the user desires. At runtime, the program loads all the libraries in a subdirectory called Entity and makes them available to the simulation.

The second example comes from a recent project in which we wanted to create a library that could load and save images of various formats. We wanted the library to be extensible, so we created a base **image_handler** class for loading and saving images.

```
class image_handler{
public:
   virtual Image loadImage(char *)=0;
   virtual int saveImage(char *, Image &)=0;
};
```

The image_handler has two public functions, used to load and save images, respectively. The **Image** class is the library's basic object type for images. It provides access to the image data and some basic image-manipulation functions.

In this case, we were not interested in creating multiple objects of each type of image_handler. Instead, we wanted one instance of each image_handler that would handle loading and saving images of that type. Rather than registering a

maker for each handler, we created a single instance of the handler in its library and registered a pointer to it with a global map. The map is no longer a factory, since it does not produce objects per se; it is more of a generic image loader/saver. The keys used here were strings representing file extensions (tif, jpg, etc.). Because a file format can have one of several different extensions (e.g., tiff, TIFF), each handler registers itself multiple times with the global map, once for each extension.

Using the library, a main program can load or save an image simply by invoking the correct handler via the file extension:

```
map <string, handler, less<string>> handler_map;
char *filename = "flower.tiff";
char ext[MAX_EXT_LEN];
howEverYouWantToParseTheExtensions(filename, ext);<\n>
// after parsing "flower.tiff" ext = "tiff"<\n>
Image img1 = handler_map[ext]->loadImage(filename);
// process data here
handler_map[ext]->saveImage(filename, img1);
```

Conclusion

Dynamic class loading allows us to create more extensible and more robust code. By using dynamic class loading combined with well-thought-out base class designs, we provide users with a practical means of extending our code.

Resources

James Norton spent most of his adult life avoiding real life by hiding out in school, first at Florida State University and then at Tulane University. The good life ended when he was awarded a Ph.D. in Electrical Engineering through what could only have been some sort of clerical error. He currently does research and systems development for Newsreal, Inc. He can be reached by e-mail at jnorton4@home.com.

Archive Index Issue Table of Contents

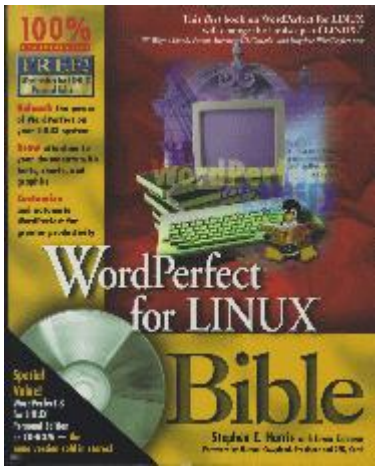
Advanced search

WordPerfect for Linux Bible

Ben Crowder

Issue #73, May 2000

This book is an excellent resource for learning what was once the most common word processor available.



- Author: Stephen E. Harris, Erwin Zijleman
- Publisher: IDG Books
- E-mail: siteemail@idgbooks.com
- URL: http://www.idgbooks.com/
- Price: $39.95 US
- ISBN: 0-7645-3374-6
- Reviewer: Ben Crowder

The *WordPerfect for Linux Bible* is an excellent resource for learning what was once the most common word processor available. Nearly seven hundred pages of instruction, very clearly written, detail almost everything you would want to know about the Linux version of WordPerfect.

A complete version—yes, you read that correctly—of WordPerfect 8 Personal Edition comes on the CD with the book, with over 5,000 clip art images and 140

fonts, as well as KDE 1.1. This is obviously a bargain, as the book and CD together are only $40 USD.

The first chapter covers installation in a reasonably thorough manner. In Chapter Two, a general overview of the WordPerfect environment is given in the form of a tutorial on writing a letter. This explains the basics of text entry in WordPerfect, the Spell-As-You-Go auto-correction feature, selecting and manipulating text, and saving your document.

We begin to delve into detail in Chapter Three, which covers the WordPerfect interface. This includes such things as getting around the screen, keystrokes, title bars and fine-tuning the document window display. In the fourth chapter, we move into formatting, with the Format-As-You-Go feature, drop caps and QuickFormatting. Chapter Five goes over WordPerfect help, including the PerfectExpert.

Part II begins with a cautionary chapter on safeguarding your work—timed backups, undo and Corel Versions are all discussed. WordPerfect actually has a miniature file manager, which is what Chapter Seven covers. In the eighth chapter, you learn about the wonderful world of Linux fonts—including the necessity of actually installing them. Adding clip art and other images to your documents is the subject of Chapter Nine, as well as inserting line art, boxes, and the WP Draw tools. Chapter Ten explains how to print your documents the way you want them printed—booklets, envelopes, print jobs and printer settings are all here.

If you want to take your document and publish it on the Web, Chapter Eleven is for you. While WordPerfect may not be the most advanced web editor (and it certainly doesn't try to be), it is useful for quick publication, and this chapter will show you how to do just that.

Since WordPerfect is predominantly used for writing actual documents, Part III centers on the tools you would use in penning concise, grammatically correct prose. Chapter Twelve helps you learn to use the spell checker, how to find that word on the tip of your tongue that is still eluding you, and how to write in French, Spanish, German, or any of the other languages that WordPerfect supports. Editing techniques—text selection, finding and replacing, bookmarks and comments—are discussed in the thirteenth chapter. In Chapter Fourteen, you learn how to get your pages numbered just right. The fifteenth chapter covers document formatting in detail: justification, paragraph formatting, styles, margins, headers, watermarks and so on. Tables and columns are explained in Chapter Sixteen.

For those budding novelists, Part IV covers working with large documents. This includes bullets and outlines (Chapter Seventeen), document references such as footnotes, lists and tables of contents (Chapter Eighteen) and multipart documents (Chapter Nineteen).

Part V centers on more advanced usage, such as templates, styles in full detail, calculations and formulas, sorting information, equations and charting data. The final section focuses on customization: the WordPerfect preferences, the toolbar and menu customization, and mass producing with labels and merge. And, of course, macros.

I would seriously recommend this book for those who are interested in WordPerfect and its Linux port. Not only does it come with the full version on CD, but it is well-written, humorous at times and easy to understand.

Ben Crowder has been heavily involved with computers for the past ten years, in almost every aspect (programming, graphics, networking, music, and just about anything else you can think of). He has been working with Linux for the past two and a half years, and has loved every second of it. In his spare time, he enjoys reading, writing, music, and tweaking things on his Linux box. He currently lives in Utah and can be reached at mlcrowd@enol.com.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

The Network Block Device

P. T. Breuer

A. Martín Lopez

Arturo García Ares

Issue #73, May 2000

A network block device (NBD) driver makes a remote resource look like a local device in Linux, allowing a cheap and safe real-time mirror to be constructed.

In April of 1997, Pavel Machek wrote the code for his Network Block Device (NBD), the vehicle for his work being the then-current 2.1.55 Linux kernel. Pavel maintained and improved the code through four subsequent releases, matched to kernels 57, 101, 111 and 132. Andrzej M. Krzysztofowicz contributed 64-bit fixes and Stephen Tweedie later contributed his substantial expertise, particularly in providing a semaphore-based locking scheme to make the code SMP-safe. We have enhanced it for use in an industrial setting by the authors, and here we describe the device, the driver and some of its development history.

The Network Block Device driver offers an access model that will become more common in this network-oriented world. It simulates a block device, such as a hard disk or hard-disk partition, on the local client, but connects across the network to a remote server that provides the real physical backing. Locally, the device looks like a disk partition, but it is a fa<\#231>cade for the remote. The remote server is a lightweight piece of daemon code providing the real access to the remote device and does not need to be running under Linux. The local operating system will be Linux and must support the Linux kernel NBD driver and a local client daemon. NBD setups are being used by us to provide real-time off-site storage and backup, but can be used to transport physical devices virtually anywhere in the world.

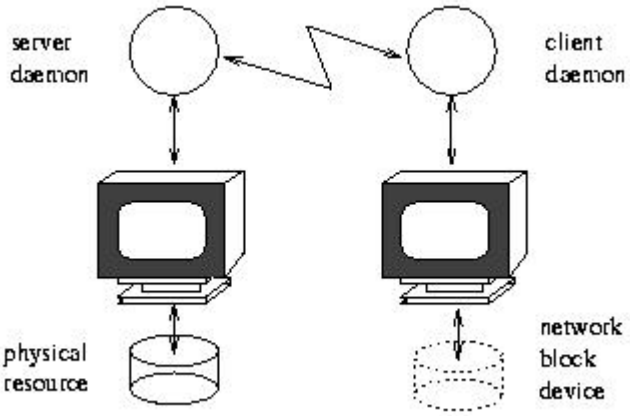


Figure 1. An NBD Presents a Remote Resource as Local to the Client

The NBD has some of the classic characteristics of a UNIX system component: it is simple, compact and versatile. File systems can be mounted on an NBD (Figure 1). NBDs can be used as components in software RAID arrays (Figure 2) and so on. Mounting a native Linux EXT2 file system over an NBD gives faster transfer rates than mounting an NFS (Network File System) from the same remote machine (see Table 1 for timings with a near-original version of Pavel's driver).

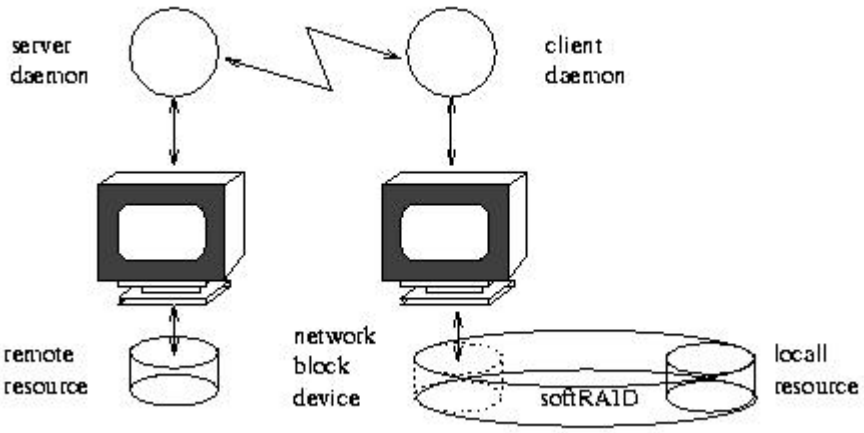


Figure 2. Remote Mirroring via Software RAID plus an NBD

Table 1

A remote Linux EXT2 file system mounted via NBD measures up to tests under default conditions like an NFS with approximately 1.5KB buffer size. That is, its apparent buffer size is the default Ethernet transmission unit size (MTU/MRU) of 1.5KB, which happens to be 1.5 times the NFS default buffer size of 1KB. The NBD gains resilience through the use of TCP instead of UDP as the transfer protocol. TCP contains its own intrinsic consistency and recovery mechanisms. It is a much heavier protocol than UDP, but the overhead in TCP is offset for NBD by the amount of retransmission and correction code that it saves.

An NBD could be used as a real-time spool mirror for a medium-sized mail server A. The failover is to a backup server B in another room, connected by a 100BT network. The NBD device connects the primary to the backup server and provides half (Y) of a RAID-1 mirror on the primary. The other half of the mirror is the primary server's own mail partition X. The composite device XY is mounted as the primary's mail spool.

On failure of A, a daemon on B detects the outage, disengages the NBD link, checks its image Y of the spool, corrects minor imperfections, mounts it locally as its mail spool and starts aliasing for A's advertised mail-exchanger IP address, thus taking over the mail service. When A recovers, it is detected and the alias is dropped. A and B then re-establish the NBD link, and the mail partition X on A is resynchronized to the NBD image Y. Then the primary server's RAID-1 device is brought back up, the NBD image is slaved to it under RAID, and the mail service resumes in the normal configuration. It is possible to make the setup symmetric, but that is too confusing to describe here.

This approach has advantages over common alternatives. One, for example, is to maintain a reserve mail server in an empty state and bring it up when the main server goes down. This makes any already-spooled mail unavailable during the outage, but reintegration is easy. Simply concatenate the corresponding files after fixing the primary. Some files on the primary may be lost.

Another method is to scatter the mail server over several machines all mounting the same spool area via NFS. However, file locking over NFS has never proven completely reliable in the past, and NFS has never yet satisfactorily been able to support the bursts of activity caused by several clients choosing to download 50MB mail folders simultaneously. The transfer rate seems to slow exponentially with folder size, and in “soft” mode, the transfer can break down under adverse network conditions. If NFS is mounted in synchronous (“hard”) mode for greater reliability while the NFS server is up, failure of the NFS server brings the clients to a halt. The NFS server has to be mirrored separately.

Table 2

A third alternative is to maintain a mirror that is updated hourly or daily. However, this approach warps the mail spool back in time during the outage, temporarily losing received mail, and makes reintegration difficult. The NBD method avoids these problems but risks importing corruption from the source file system into the mirror, when the source goes down. This is because NBD operations are journaled at the block level, not the file system level, so a complete NBD operation may represent only a partially complete file operation.

The corruption and subsequent repair is not worse than on the source file system if the source actually crashed; if connectivity was the only thing lost, the source system may be in better shape at reintegration than the mirror. To avoid this problem, the Linux virtual file system (VFS) layer must be altered to tag block requests that result from the same file operation, and the NBD must journal these as a unit. That implies changes in the VFS layer, which we have not yet attempted to implement.

Given the slightly chequered history of NFS in Linux, it may be deemed something of an advantage that the NBD driver requires no NFS code in order to supply a networked file system. (The speed of NFS in the 2.2.*x* kernels is markedly superior to that of the 2.0.*x* implementations, perhaps by a factor of two, and it no longer seems to suffer from nonlinear slowdown with respect to size of transfer.) The driver has no file system code at all. An NBD can be mounted as native EXT2, FAT, XFS or however the remote resource happens to be formatted. It benefits in performance from the buffering present in the Linux block device layers. If the server is serving from an asynchronous file system and not a raw physical device at the other end, benefits from buffering accrue at both ends of the connection. Buffering speeds up reads one hundred times in streaming measurements (it reads the same source twice) depending on conditions such as read-ahead and CPU loading, and appears to speed up writes approximately two times. With our experimental drivers, we see raw writes in “normal use” achieving about 5MBps to 6MBps over switched duplex 100BT networks through 3c905 network cards. (The quoted 5-6MBps is achieved with buffering at both ends and transfers of about 16MB or 25% of installed RAM, so that buffering is effective but not dominant.)

In 1998, one of the authors (Breuer) back-ported the then-2.1.132 NBD to the stable 2.0-series kernels, first taking the 2.1.55 code and reverting the kernel interface, then paralleling incremental changes in Pavel's subsequent releases up to 2.1.132. That code is available from Pavel's NBD web pages (atrey.karlin.mff.cuni.cz/~pavel/nbd/nbd.html) along with Pavel's incrementals. The initial back port took out the new kernel dcache indirections and changed the networking calls back to the older style. The final changes paralleled late 64-bit adaptations in Pavel's sources.

Like the original, the back-ported code consists of a kernel module plus a small adaptation to the kernel block-driver interface, and user-space server and client daemons. Pavel proved to be an extremely helpful correspondent.

The driver worked very well on the development Linux machines in use in our department, where we maintained a much-modified 2.0.25 kernel code base for several years. With the perceived robustness of the 2.0.36 kernel release in

particular, we ported the driver forward. Surprisingly, it failed completely. It locked up any client after only about 0.5MB of transfers.

To this day, we do not know precisely the nature of that problem. Stephen Tweedie examined the algorithm in the ported driver and concluded it to be the same as the 2.1.132 algorithm, which works, and Pavel approved the protocols. Tracing back the 2.0-series kernels patch by patch failed to find any point at which the driver stopped working. It did not work in any standard release. Our best guess is that nonstandard scheduler changes in our code base were responsible for the initial port working smoothly in our development environment, and that general changes in the 2.1 series had somehow obviated the problem there.

Experiments showed that the in-kernel queue of outstanding block requests grew to about 17 or 18 in length, then corrupted important pointers. We tried three or four methods aimed at keeping the queue size down, and empirically seemed to control the problem with each of them. The first method (Garcia) moved the networking out of the kernel to the user-space daemons, where presumably the standard scheduling policy served to remove an unknown race condition. The second method (Breuer) kept the networking in-kernel, but serialized network transfers so that every addition to the pending queue was immediately followed by a deletion, thus keeping the queue size always equal to zero or one. The third method (Garcia) removed the queue mechanism entirely, keeping its length at zero. A fourth method (Breuer) made the client daemon thread return to user space after each transfer to allow itself to be rescheduled.

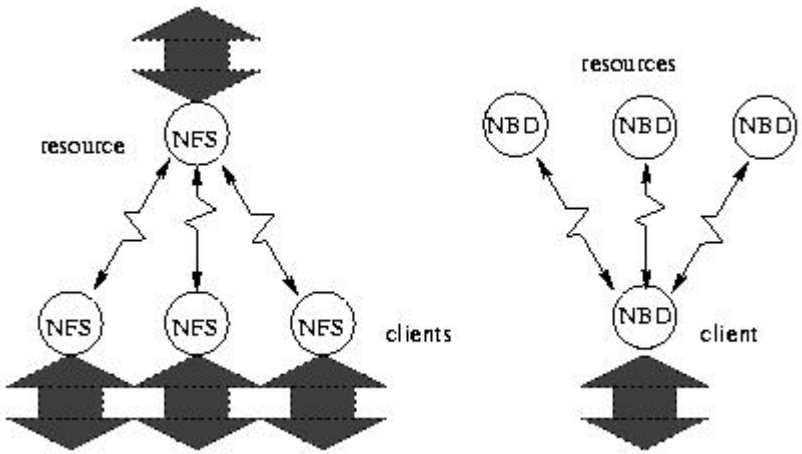


Figure 3.

NFS provides a star configuration, with the server at center (left). NBD, with the help of internal or external RAID, provides an inverted topology with multiple servers and a single client.

For a considerable time, running in serialized mode seemed to be the best solution, although the user-space networking solution is also reliable. With our recent development work, we have gained deeper insight into the kernel mechanisms working here, and although no definitive explanation of the original instability is available, we do have completely stable drivers working in a fully asynchronous in-kernel networking mode as well as user-space versions of the same protocols. A suggestion is the possible misgeneration of code by the gcc compiler (**a>b=0; c>d=a; if(c>d>b) printk("bug"**); output generated just before lockups!) but it is possible that stack corruption caused a misjump into the linear code sequence from elsewhere. Increased re-entrancy of the driver code and more forgiving network transfer protocols definitely increased the stability while the problem was detectable, which is a good co-indicator for a race condition.

Setting up a Connection

Steps in Setting Up an NBD-Based File System

The five steps required to create a file system mounted on a network remote device are outlined in Figure 4. For example, the following sequence of commands creates an approximately 160MB file on a local file system on the server, then launches the server to serve from it on port 1077. Steps 1 and 2 are

```
dd if=/dev/zero of=/mnt/remote bs=1024 count 16000
nbd-server 1077 /mnt/remote
```

On the client side, the driver module must be loaded into the kernel, and the client daemon started. The client daemon needs the server machine address (192.168.1.2), port number and the name of the special device file that will be the NBD. In the original driver, this is called /dev/nd0. Step 3 is

```
insmod nbd.o
nbd=client 192.168.1.2 1077 /dev/nd0
```

A file system can then be created on the NBD, and the system mounted locally with Steps 4 and 5:

```
mke2fs /dev/nd0
mount -text2 /dev/nd0 /mnt
```

In our current drivers, multiple ports and addresses are allowed, causing redundant connections to be initiated. Here, the server offers several ports instead of one:

```
dd if=/dev/zero of=/mnt/remote bs=1024 count=16000
nbd-server 1077 1078 1079 1080 /mnt/remote
```

The current client can use all these ports to the server, and here we direct two of them to a second IP (192.168.2.2) on the server so that we can route through a second network card on both machines and thus double the available bandwidth through our switched network.

```
insmod nbd.o
nbd-client 192.168.1.2 1077 1078 192.168.2.2 1079 1080 /dev/nda
```

In the current drivers, the NBD presents itself as a partitioned block device nda, although the “partitions” are not used in a standard way. Their device files nda1, nda2 and so on are used as kernel communication channels by the subordinate client daemons. They provide the redundancy and increased bandwidth in the device. The whole-device file nda is the only one that accepts the standard block-device operations.

Driver and Protocol Overview

On insertion of the kernel module, the driver registers with the kernel. As the client daemon connects for the first time to its server counterpart, the original driver hands the file descriptor of the socket to the kernel . Kernel traces the descriptor back to the internal kernel socket structure and registers the memory address in its own internal structures for subsequent use. Our current drivers keep the networking in user space and do not register the socket.

The client daemons and server daemons then perform a handshake routine. No other setup is required, but the handshake may establish an SSL channel in the current generation of drivers, which requires SSL certificates and requests to be set up beforehand.

Pavel's original driver code comprised two major threads within the kernel. The “client” thread belongs to the client daemon. The daemon's job is to initiate the network connection with the server daemon on the remote machine, and hand down to the kernel via an **ioctl** call the socket it has opened. The client daemon then sits blocked user-side in an **ioctl** call while its thread of execution continues forevermore within the kernel. It loops continuously transferring data across the network socket from within the kernel. Terminating the daemon requires terminating the socket too, or the client daemon will remain stuck in the loop inside the kernel ioctl.

A “kernel” thread enters the driver sporadically as a result of pressures on the local machine. Imagine that **echo hello >! /dev/nd0** is executed (the block-device names for the original driver are nd0-127, and they take major number 43). The **echo** process will enter the kernel through the block device layers, culminating in a call to the registered block-device request handler for a write to the device. The kernel handler for NBD is the function **nbd_request**. Like all block-device request handlers, nbd_request performs a continuous loop

while(req = CURRENT), **CURRENT** being the kernel macro that expands to the address of the write request struct. After treating the request, the driver moves the pointer on with **CURRENT = CURRENT->next** and loops.

The kernel thread's task is to do the following:

- Link the request **req = CURRENT to** the front of the pending transfer list.
- Embed a unique identifier and emit a copy across the network to the server daemon at the other side of the network socket.

The unique identifier is the memory address of the request **req**. It is unique only while the request has not yet been serviced, but that is good enough. (When the driver used to crash through the mysterious corruption we were never able to pin down, the crash was often associated with duplicated entries and a consequently circular list, which may be a clue.)

On the other side of the network, the server daemon receives a write request, writes “hello” to its local resource, and transmits an acknowledgement to the client containing the unique identifier of the request.

The client daemon thread on the local machine is in its loop, blocked inside the kernel on a read from the socket, waiting for data to appear. Its task is now to do the following:

- Recognize the unique identifier in the acknowledgement, comparing it with the oldest (last) element req in the linked list of partially completed requests.
- Unlink the request req from the list of incompletes, and tell the block layers to discard the structure via a call to **end_request**.

This protocol requires that the acknowledgement received be for the request pending on the tail of the driver's internal list, while new requests from the kernel are added to the head. TCP can guarantee this because of the sequential nature of the TCP stream. Even a single missed packet will break the current driver, but it will also mean the TCP socket is broken. The socket will return an error in this situation. That error message allows the driver to disengage gracefully.

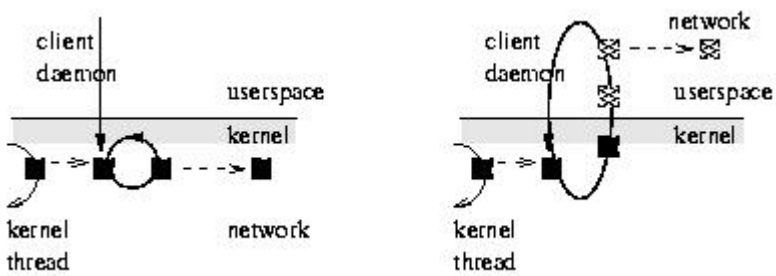


Figure 4.

Kernel networking vs. user-space networking in an NBD. User-space networking requires an extra copy and other overheads, but affords much greater flexibility. The overhead can be offset by transferring multiple requests at a time.

The client-side control flow in the original kernel driver is shown schematically on the left side of Figure 4. The black rectangle represents a request. It is linked into the device's request queue by a kernel thread and is then swept up in the client daemon's perpetual loop within the kernel. The client thread performs networking within the kernel. In the drivers we have subsequently developed, we have come to favour user-side networking, in which the client thread deals user-side with a copy of the request transferred from within the kernel. It dives repeatedly into the kernel to copy across the data, then transmits it in standard network code. The overheads are much greater, but the flexibility is also much greater. The overhead can be ameliorated by transferring multiple requests across at a time, and our current drivers do this. Normally, 10 to 20 requests of one block each will be transferred in each visit to the kernel. The cost of copying between kernel and user space cannot, however, be avoided. Multiple client daemons contend for the kernel requests as the clients become free, transferring them across the network through possibly distinct routes and physical devices. The situation is depicted in Figure 4. Each client daemon handles one channel, but will mediate any request. The channels provide redundancy, resilience and bandwidth.

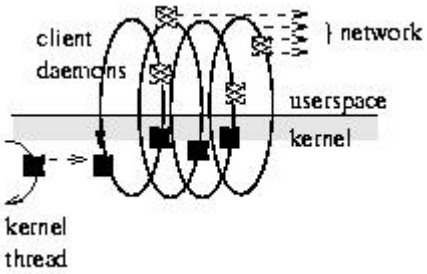


Figure 5.

Multiple client daemons capture kernel requests in the current NBD drivers, providing redundancy and load balancing through demand multiplexing across several distinct network channels.

Table 3

The complete data protocol sequence “on the wire” is shown in Table 3. Note that the unique ID is 64-bit, so it may use the request's memory address as the identifier on a 64-bit architecture. Curiously, the requested data offset and

length are 32-bit byte offsets in the original driver, although they are calculated from sector numbers (sectors are 512 bits each) which might well have been used instead. This is a hidden 32-bit limitation in the original NBD. Our versions implement 64-bit limits on a 32-bit file system or machine architecture. The server daemon has been modified to multiplex requests beyond 32 bits among several distinct resource files or devices.

Improvements

Table 4

The goal of our industrial partners in the driver development was to obtain improvements in areas which are important to an industrial-strength setting (see Table 4).

For example, the original driver broke irretrievably when the network link broke, or when a server or client daemon died. Even if reconnected, an I/O error became visible to the higher layers of, for example, an encompassing RAID stack, which resulted in the NBD component being faulted off-line. We have worked to increase robustness by making externally observable failure harder to induce, letting the driver deal internally with any transient problems. The current drivers log transactions at the block level. A spoiled transaction is “rolled back” and resubmitted by the driver. Doubly completed transactions are not considered an error, the second completion being silently discarded.

A transaction spoils if it takes too long to complete (“request aging”), if its header or data becomes visibly corrupted or if the communication channel errors out between initiation and completion of the transaction. To keep their image of the link state up to date, the drivers send short keep-alives in quiet intervals. Data can still be lost by hitting the power switch, but not by breaking the network and subsequently reconnecting it, nor by taking down only one or two of the several communication channels in the current drivers. The drivers have designed-in redundancy. Redundancy here means the immediate failover of a channel to a reserve and provision for the multiple transmission and reception of the same request.

There are many ways in which connectivity can be lost, but differently routed network channels survive at least some of them. The daemon client threads (one per channel) demand-multiplex the communications among themselves according to demand and their own availability (kernel-based round-robin multiplexing has also been tried). Under demand-multiplexing, the fastest working client threads bid for and get most of the work, and a stopped or non-working thread defaults its share of the workload to the working threads. In conjunction with journaled transactions and request aging, this seems very effective. Channels automatically “failover” to a working alternative.

Speed is also an important selling point in industry. Our client-daemon threads are fully asynchronous for maximum speed. They are not synchronized via locks or semaphores inside the kernel (apart from the one semaphore atomicising accesses to the NBD major's request queue). This means kernel requests may be treated out of order at various stages of their lifetime, so the driver has had to be revised carefully to meet this requirement. In particular, the standard Linux kernel singly linked request list has been converted to a doubly linked list, appropriating an otherwise unused pointer field. The driver can then “walk the queue” more easily to search for a match in the out-of-order case. The kernel really should implement a doubly linked request-queue structure. The main brake on speed is probably the indiscriminate wakeup of all client threads on arrival of one kernel request, but when the requests come in faster than the daemons can send them, that overhead disappears.

Although failover of a network channel may be unobserved from outside the driver, it will degrade communications in some senses of the word. So, our current daemon implementations reconnect and renegotiate in an attempt to restore a failed channel. If the daemons themselves die, then they are automatically restarted by a guardian daemon, and the kernel driver will allow them to replace the missing socket connection once the authentication protocol is repeated successfully. Upon reconnection, any pending read/write requests unblock if they have not already been rolled back and resubmitted via another channel. If the device is part of a RAID stack, the outage is never noticed.

Connections between remote and local machines, when multiplexed over several ports as in our drivers, can be physically routed across two different network interfaces, doubling the available bandwidth. Routing across *n* NICs multiplies available bandwidth by *n* until the CPU is saturated. Since streaming through NBD on a single 3c905 NIC loads a Pentium 200MHz CPU to about 15% on a 100BT network, a factor of about four to five in increased bandwidth over a single NIC may be available (we have not been able to test for the saturation level, lacking sufficient PCI slots to do so).

Another important industrial requirement is absolute security of the communication channels. We have chosen to pass all communication through an external SSL connection, as it hives off the security aspect to the SSL implementation which we trust. That decision required us to move the networking code entirely out of the kernel and into the client daemons, which are linked with the openSSL code. Moving the code user-side slows the protocol, and it is slowed still further by the SSL layers. The fastest (non-trivial) SSL algorithm drops throughput by 50%, the slowest by 80%. An alternative is to pass the communications link in-kernel through a **cipe** tunnel, or to site

either the server or client daemon on an encrypted file system, but we have not experimented with these possibilities.

SSL offers us the mechanism with the best understood security implications and we have plumped for it. As fallback, a primitive authentication protocol is provided in the driver itself—tokens are exchanged on first connection, and reconnects require the tokens to be presented again. The daemons wrap this exchange and the whole session in the openSSL authentication mechanisms, so it is hardly ever required. Compressing the data as it passes across the link would also provide greater bandwidth (as well as perhaps security), but so far we have not implemented it. It can be applied in the user-side daemon codes.

Whatever limits may or may not be present in the rest of the code, the server daemons are constrained on 32-bit architectures by the 32-bit implementation of EXT2 and other file systems. They can seek up to only a 31-bit (32-bit) offset, which limits the size of the served resource to 2GB (4GB). Our server daemons make RAID linear or striping mode available, however. In these modes, they serve from two or more physical resources, each of which may be up to 2GB in size, to make a total resource size available that breaks the 32-bit barrier by any amount required. Striping seems particularly effective in terms of increased bandwidth, for unknown reasons. The daemons can also serve directly from a raw block device. In those circumstances, it is not known to us if there is a 32-bit limitation in the kernel code—presumably not.

Listing 1

A /proc/nbdinfo entry provides us with details of the driver state, the number of requests handled, errored and so on. The page is read-only (see Listing 1).

Adjustments to default driver parameters, such as the interval between keep-alives and the number of blocks read-ahead, are made via module options on loading the driver. There is currently no mechanism to change parameters at other times. A writable \proc entry or a serial control device is probably desirable. Some degree of control is vested in the client guardian daemon and subordinate client daemons, which send special ioctls to the device on receipt of particular signals. The USR2 signal, for example, triggers an ioctl that errors out all remaining requests, ejects client-daemon threads from the kernel and puts the driver in a state in which the module code can be safely removed from the running kernel. It is, of course, a mechanism intended to aid debugging. The disadvantage of this kind of approach is that it requires a client daemon to be running before control can be exercised. In the future, we intend to implement the /proc-based writable interface. The use of fake partition information in the device is also an appealing route towards obtaining better reports and increased control.

Writing Linux Device Driver Code

Writing the driver code has been a salutary experience for all involved. The best advice to anyone contemplating writing kernel code is—don't. If you must, write as little as possible and make it as independent of anything else as possible.

Implementing one's own design is relatively easy as long as things go well. The very first bug, however, reveals the difficulty. Kernel code bugs crash the machine often, giving scant opportunity to detect and correct them. Twenty reboot cycles per day is probably near everyone's limit. On occasion, we have had to find a bug by halving the code changes between versions until the precise line was located. Since a moderate number of changes can lead to a patch of (say) 200 lines or more, eight recompilation cycles and tests might be required to locate the point change involved. That says nothing of the intellectual effort involved in separating the patch into independent parts in order to be able to recompile and the effort involved in developing a test for the bug or identifying the behavioural anomaly in the first place. Between one and two weeks is a reasonable estimate for locating a bug via code-halving.

It is very important to have an always-working kernel code. It doesn't matter if the code does not have the right functionality, but it must do what it does right. The code development must be planned to move forward in stages from one working state to another. There must exist no stage of development in which the driver does not work, such as for example having altered a protocol and not yet balanced the change with corresponding changes elsewhere.

Having a working version implies checking in working versions regularly (we used CVS). Check-in occurs only on working versions. On a couple of occasions, we had to fork the line to allow two development areas to proceed independently (moving the networking code out of kernel while reworking the reconnection protocols, for example), then reintegrate the changes via a sequence of non-working minor revisions, but we always had a previous working version available which we tried to make minimal changes to.

Debugging techniques essentially consist of generating usable traces via **printk** statements. We had printks at entry and exit of every function, activity and branch. That helps us discover where the coding bug occurs. Often, however, the bug is not detectable from the code trace, but rather must be inferred through behavioural analysis. We had a serious bug that was present through half the development cycle and was never detected until integration tests began. It was completely masked by normal kernel block-buffering and showed up as apparent buffer corruption only in large (over 64MB) memory transfers. An **md5sum** of the whole device would sometimes return differing results when the rest of the machine was under heavy load. It turned out to be two simple bugs, one kernel-side and one server-side, that had nothing to do with

buffering at all. In this kind of situation, brainstorming possible causal chains and devising tests for them, then running the tests and interpreting the results is the only feasible and correct debugging technique. This is the “scientific method” as expounded in the 18th and 19th century, and it works for debugging.

Kernel coding really begins to bite back when kernel mechanisms not of one's own devising have to be assimilated. Interactions with the buffering code had to be taken somewhat on trust, for example, because reading the buffering code (buffer.c) does not tell the whole story in itself (for example, when and how buffers are freed by a separate kernel thread). It is good advice to try and limit interactions with the other kernel mechanisms to those that are absolutely predictable, if necessary, by patterning the interactions on other driver examples. In the case of the NBD driver, the original was developed from the loopback driver (lo.c), and the latter served as a useful reference throughout.

Summary

The Network Block Device connects a client to a remote server across a network, creating a local block device that is physically remote. The driver we have developed provides mechanisms for redundancy, reliability and security that enable its use as a real-time backup storage medium in an industrial setting as well as allowing for other more imaginative modes. A mobile agent that takes its home environment with it to every system it visits, perhaps? In terms of speed, an NBD supporting an EXT2 file system competes well with NFS.

Acknowledgements

P. T. Breuer, (ptb@it.uc3m.es)

A. Martín Lopez

Arturo García Ares

Archive Index Issue Table of Contents

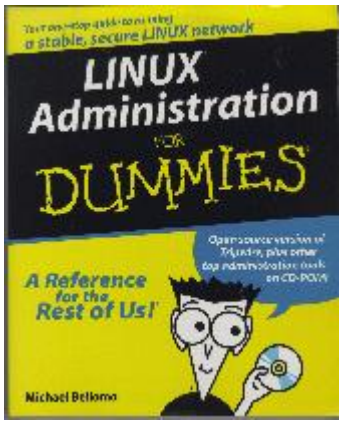
Advanced search

LINUX Administration for Dummies

Harvey Friedman

Issue #73, May 2000

The best feature of the book is that several times the author refers to Linux for Dummies, 2nd Edition to find basic information.



- Author: Michael Bellomo
- Publisher: IDG Books Worldwide, Inc.
- E-mail: siteemail@idgbooks.com
- URL: http://www.idgbooks.com/
- Price: $24.99 US
- ISBN: 0-7645-0589-0
- Reviewer: Harvey Friedman

Linux administration? What is this book really about? Was it system administration similar to the classic “UNIX System Administration Handbook” by Nemeth et al., except with particular emphasis on Linux? Was it network administration dealing with hubs, switches, bridges, routers, etc.? What was on the CD-ROM, and how did it apply to administration?

After reading this book, I'm not sure I know any answers. In fact, I think the best feature of the book is that several times the author refers to *Linux for*

Dummies, 2nd Edition to find basic information. The book lacks organization, the highlighted examples often have the wrong fonts for the computer/user dialog, and sections often contain contradictory paragraphs. In general, the writing style is obnoxious.

It would appear that the author either never heard of (or never used) the commands **more** or **less** except in the MS-DOS piping sense. In a few instances, despite admonishing the reader not to change anything in the file they want to view (as part of the example), he encourages using vi to view, despite warning that it may be tricky to use. It is just too easy for the beginner to err when using vi as root on a live system.

The book is concerned about users on a “Linux system”, but doesn't say *how* they may be there. Nothing about, for example, Cyclades multi-port cards to allow multiple simultaneous users, or even the suggestion that family members take turns at the single-seat console. Just how is the na<\#239>ve user with a desktop computer going to know what is meant by a “Linux terminal”? Though he does mention dual-boot on one page, it is more to confuse the reader than to explain it. He was correct in stating that one can dual-boot via LILO; he missed the opportunity to suggest that, in addition, one could boot from a floppy disk, via loadlin from MS-DOS, from a CD-ROM, etc.

So from the limited discussion, I guess the reader must assume one is attached to the Ethernet for multiple users. The discussion of Ethernet cabling is lacking, in particular there is no mention of a hub when describing 10-BaseT. The author leads the reader to assume that **ping** stops automatically, but on Red Hat Linux 5.2 or 6, one enters **CTRL-C** after a reasonable time to get ping statistics about network connections.

Chapters 8 and 9 offer simple introductions to the basic protocols: TCP and UDP, and Internet addresses, masks and services. However, the explanation and example of subnets is wrong.

Chapter 11 considers UUCP and FTP quite superficially. That is not a problem with UUCP, because few people use it these days. But, when discussing setting up an anonymous FTP server, the author lists what purports to be a step-by-step method. Unfortunately, he doesn't describe what should be in /ftp/bin nor whether an /ftp/lib directory is needed. Why would technical book authors not try out their examples?

A whole chapter (12) was devoted to SLIP, but then the author claims it has basically been superceded by PPP, which is treated in a too-brief chapter (13). The chapters in Part V on Network File and Machine Sharing appear to have been written about five years ago, particularly those on NFS, NIS, and DNS.

Chapter 18, on printing, seems somewhat useful. I will note that if one wants to print on a printer on an Apple network, it is not always necessary to do so through Netatalk. Newer Apple Macintosh Laserwriters can be assigned their own IP numbers and names and can be accessed by following the instructions in the manual for printing from a networked UNIX machine.

Part VI, Electronic Mail, News and Web Browsing, is brief but somewhat better than most of the other parts. I was surprised that the author didn't suggest creating a message file to someone, using vi, and then giving the command “mail someone@domain < message”. I guess he had never used the “mail” program when it was the only choice before such programs as Elm, Pine, and Netscape were written. I agree with his choice of Pine for e-mail and Usenet news, and Netscape for web browsing.

Part VII, Network Security, is mostly just buzzwords. For example, let me quote a couple of sentences. “TCFS, the Transparent Cryptographic File System, is a newer version of CFS with greater capabilities. Its chief advantage is that TCFS is user transparent, which is the same attribute that makes CFS so easy to use and administer.” OK, this tells us that it is newer, but what are the “greater capabilities”? The accompanying CD-ROM dataplate contains such programs as Tripwire, Mon, Sudo, etc., but newer versions or better replacement tools can be downloaded.

Part VIII, Linux Disasters and Recovery Techniques, has a few commonsense ideas. One is to make regular backups. Others are to keep a good set of Linux reference material around, read Linux newsgroups, and make friends with other administrators.

Part IX is the “for Dummies” Part and lists some utilities, tools and web sites. Since the author seemed so inept through most of this book, I have no reason to think his personal ratings and recommendations are useful. I will note that I do think this part was written more recently than many other chapters in his book.

Overall, I think the bulk of this book was a reworked version of a UNIX system administration book from roughly five years ago. For someone who has never been previously exposed to system administration or network administration, looking up buzzwords from this book on a web browser might actually be helpful. Anyone who set up their Linux box following *Linux for Dummies, 2nd Edition*, or *Linux for the Compleat Idiot*, or *The No BS Guide to Linux* will probably be disappointed if they move on to this book. I don't understand how two reviewers on Amazon.com could give this book four stars out of five. I'd give it one star out of five, at best.

Harvey Friedman can be reached via e-mail at fnharvey@u.washington.edu.

Archive Index Issue Table of Contents

Advanced search

A Real-Time Data Plotting Program

David Watt

Issue #73, May 2000

How to program using the Qt windowing system in X.

This article describes the implementation of rtp (real-time plotter), a live x,y data plotting utility based on the Qt windowing library. **rtp** combines live updates with zoom in, auto-scaling, and auto-tracking modes. It is meant to be used where **gnuplot** is limited, such as the termination of a live data pipeline. However, rtp is small and does not attempt to cover gnuplot's large feature set for producing publishable data plots.

The rtp source code is released under the GPL and is available at metalab.unc.edu/pub/linux/science/visualization/rtp-1.0.0.tar.gz. I developed and tested it under Red Hat 6.0, with Qt 1.44. A README file is included in the package to help you build and use rtp. A screenshot of rtp is shown in Figure 1.

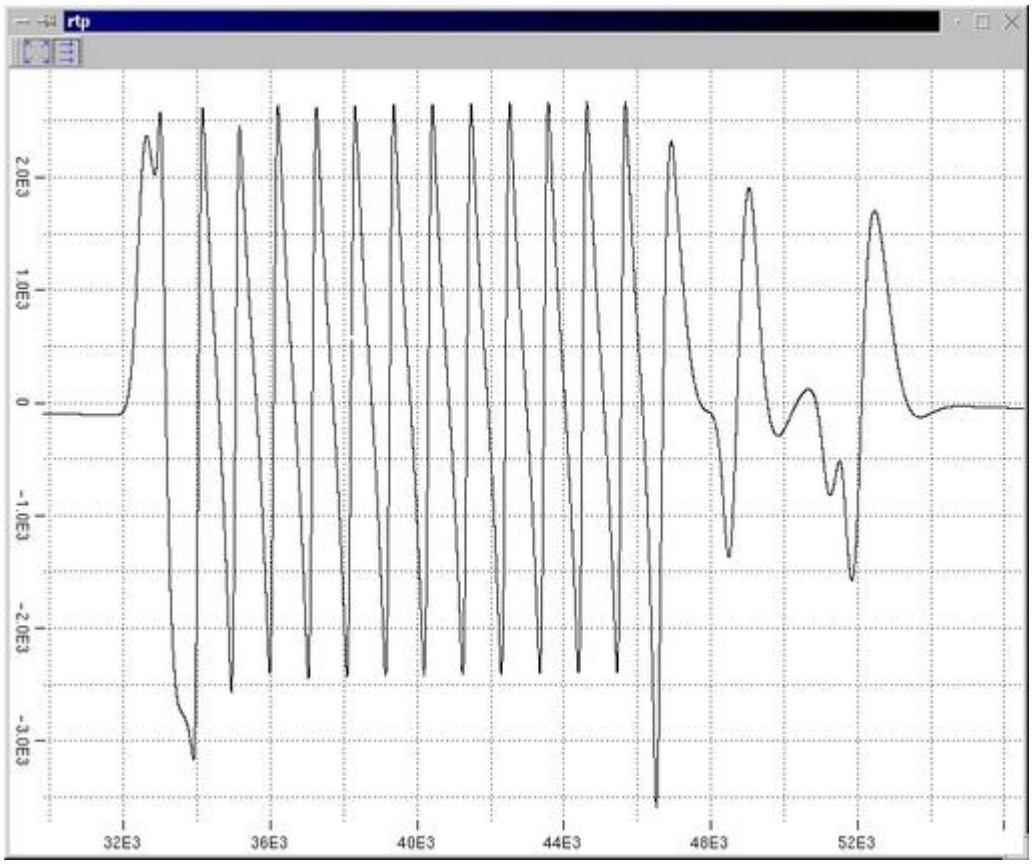


Figure 1. Rtp Screenshot

rtp provides real-time updates and basic mouse-driven resolution selection. However, it lacks gnuplot's ability to send formatted, titled plots to a printer. **rtp** is still a simple piece of software (1200 lines of code) that needs many features added. By describing its principles here, I hope to provide a useful, gentle example of an application based on the Qt library and the X Window System. I also hope to motivate some interested people to do more work on a Linux-based real-time, interactive data visualization system. This could be done either by extending rtp or as a completely new project.

Viewing Modes

Because all of rtp's data comes from STDIN (standard input), interaction with the user through the X Window System is limited to setting the viewing mode. It allows the user to change the viewing mode even as new data points are being processed. The viewing modes are as follows:

- Auto scale: the scaling is adjusted when necessary to include all received data points. This is the default mode and can be selected by pressing a button on the toolbar.
- Auto tracking: maintains a fixed scaling, but varies the viewport offset to track the latest points. This mode is selected by pressing a button on the toolbar. The scaling will be fixed at what it was before the toolbar button was pressed.

- User-defined fixed: maintains a fixed viewport (both scale and offset), as defined by the user. This mode is selected when the user drags out a viewport in the plot window with the mouse.

Qt Library

I based rtp on the Qt library, because many others in the Linux community are using it (e.g., KDE) and because of its high-quality documentation. An HTML tree (guaranteed to be synchronized with the Qt source because it is automatically generated from the source code and comments) describes all of Qt's classes and functions. Dalheimer also wrote a book on Qt programming that is a very helpful introduction (see Resources).

The Qt library provides a GUI programming environment that is quite complete. When programming in the Qt environment, no reference to the underlying XLib library is necessary. Qt's functionality extends beyond the GUI domain to include container classes that implement several standard data structures.

Each of Qt's functional components is packaged as a C++ class, giving C++ wizards much to ponder and tinker with and those of us who like to write operational code a good tool set. For myself, having about a year of experience writing production C code with only a college course in C++, it was fairly easy to learn the Qt C++ framework.

The Qt library makes integration of independently developed classes easier through its C++ extensions: “signals” and “slots”. A signal is a class member function that is undefined at compile time. A slot is a member function that is specially designated for connection to a signal at runtime. For example, a GUI button class could have a **Push** signal. At runtime, a plot window's slot **Render** could be connected to the button's **Push**. From then on, code that calls the button's **Push** method effectively calls the plot window's **Render** method.

Code based on the signals and slots mechanism is easier to read and maintain than that dealing with runtime function-pointer tables. (I'd bet the implementation uses a function pointer or two.) Qt also takes care of annoyances such as stubbing non-connected signals to an empty function, so you don't get a segmentation fault from a null pointer.

The drawback of signals and slots is that they are non-standard C++ extensions using new syntax, so Qt code with signals and slots must be passed through a preprocessor provided with the Qt library before it can be compiled. Dalheimer's book explains signals and slots in sufficient detail for you to start using them.

Architectural Requirements

In order to provide an acceptable user interface, rtp must quickly respond to GUI events (i.e., mouse events, etc.) at all times. This requirement would be easily met if all program activity were directed by GUI events. For example, an interactive drawing program is entirely GUI-driven, so its only responsibility is to execute relatively short sequences of code in response to GUI events.

rtp's architecture is complicated by two additional requirements, beyond the snappy GUI response. It must quickly update when new data points are available on STDIN. This feature is what differentiates rtp from other plot programs, such as gnuplot. It must also deal with the fact that rendering the data set often takes more time than is acceptable for a GUI delay. This precludes the use of a simple function call to render the whole plot.

Multiplexing Tasks

Fundamentally, there are three “tasks” that rtp must multiplex, listed below from highest to lowest priority:

1. Respond quickly to GUI events. These events come as data from the X server on a socket.
2. Read data points from STDIN as they become available.
3. Render the data set into the plot window when it needs updating.

The Qt library provides mechanisms that support this processing structure. The first mechanism is the QSocketNotifier class. When we create a QSocketNotifier object, we pass it the file descriptor of interest. (The fancy name QSocketNotifier made me think the class was all tied in with network sockets, when in reality it can work with most any file descriptor.) In the case of rtp, this is the STDIN file descriptor (**STDIN_FILENO**). We then connect QSocketNotifier's **activated** signal to a particular slot that deals with new data.

The second mechanism is the QTimer class. This class is provided to support regularly scheduled background processing, as well as single-shot timed events. The Qt documentation tells us that by setting up a QTimer object with zero timeout, we can cause a function to be called whenever there are no events to process. Again, the mechanisms for connecting the QTimer to the actual function that does the background processing are signals and slots.

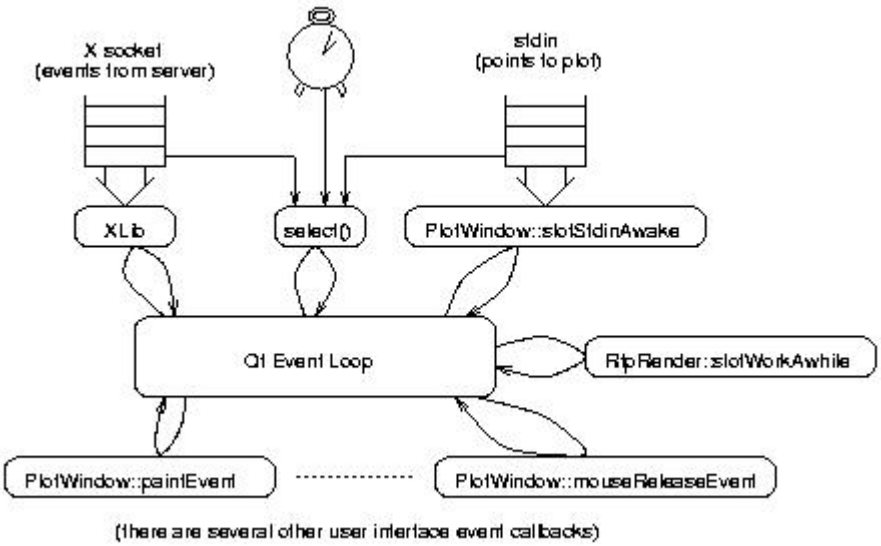


Figure 2. rtp Control Flow and Data Processing

Components

Figure 2 illustrates rtp's control flow and data processing scheme. The Qt event loop is the control center for the application. It calls functions in the rtp application as events occur. Each arrow in the diagram represents a call into a function or library. Note that only the functions which have names starting with **PlotWindow** or **RtpRender** are actual rtp code. The rtp functions consist of X event callbacks (such as **PlotWindow::paintEvent** and friends), the QSocketNotifier callback (**PlotWindow::slotStdinAwake**) and the QTimer callback (**RtpRender::slotWorkAwhile**).

XLib is the lowest-level C library provided as an interface to an X server across a byte-stream socket. It manages both the input side of the socket, which provides events, and the output side which sends requests to the server. (Note that Figure 2 shows only the input side.) XLib also provides certain performance optimizations, such as filtering redundant events and delaying requests in an internal queue in order to group requests in large data blocks for efficient socket usage. For details, see Adrian Nye's classic XLib book in Resources.

The POSIX **select** system call is commonly used by applications that service more than one file descriptor (socket) in a single thread. **select** is used by applications, such as rtp, that must respond to data on any of several file descriptors and do not wish to waste CPU time by polling. Additionally, Qt uses select's timeout function to start QTimer-scheduled functions.

The select call in the Qt library is the only place (to my knowledge) where the rtp process can block. For newcomers to systems programming, I should explain what “blocking” means. A multitasking operating system such as Linux must be able to multiplex the execution of a large number of programs on a

smaller number of processors. By trapping interrupts, Linux switches the processor(s) among running programs in a certain order. This makes it impossible for a single program to lock the system by entering an infinite loop.

Because Linux has pre-emptive multitasking, rtp could enter an infinite loop waiting for either an X event or a data point on STDIN without locking the system. However, the CPU time spent in this loop would unnecessarily degrade the performance of the rest of the executing programs. Therefore, most calls into the kernel for I/O will "block" the executing program until the I/O is complete. The program will be removed from the set of programs run by the CPU. Once the I/O is done, the program is marked as "runnable" and will re-enter the kernel's run queue to be switched in and out of the CPU along with other runnable programs.

select is Qt's way of saying, "I have nothing to do until an X event is available, an I/O event occurs on one of the QSocketNotifier objects, or a timeout from one of the **QTimer** objects expires." From Qt's point of view, select waits for one of these events to occur, then returns.

The Qt documentation clearly describes how to use QSocketNotifier and QTimer to hook into select. However, it does not fully describe the priority levels of X events vs. other socket events vs. timer events. In writing a program such as rtp, it is important to understand these details, because the program's performance greatly depends on them.

Inside Qt's Event Handling

In order to understand how the X socket, other sockets and timers are prioritized, we have to look into the Qt source. Troll makes the Qt source freely available (see Resources for the URL). The code we want is in /src/kernel/ qapplication_x11.cpp, under the distribution tree. Note that while Qt source is freely redistributable, Troll's license prohibits modification, unlike the GPL.

The function **QApplication::processNextEvent**, which is called by the main event loop, services the X socket, QSocketNotifier sockets and QTimer timers. QApplication::processNextEvent first checks for an X event to process. If none is found, it enters the select system call.

After select returns, QApplication::processNextEvent dispatches events to all QSocketNotifier objects whose file descriptors are ready. It then dispatches events to all QTimers whose timeouts have been reached. The event loop for Qt 1.44 can be summarized as follows (fd stands for file descriptor):

```
while (1)
{
   if (X event pending) {
```

```
        dispatch X event;
        continue;
    }
        timeout = minimum of all QTimer times to
           next event;
        select (X fd, all QSocketNotifier fd's,
           timeout);
        dispatch events to all QSocketNotifier's with
          active sockets;
        dispatch events to all QTimer's with expired
           times;
}
```

Note that X events get highest priority, in the sense that as long as there are more X events, the loop will ignore the QSocketNotifier's and QTimer's events. However, when an X event is not available, it is possible that Qt will execute *every* QSocketNotifier and QTimer event before returning to X events. This means we must consider the sum of registered QSocketNotifier and QTimer event processing times as the worst-case user interface latency.

Data Structures

From here on, I will refer to the rtp code in some detail. You may want to download the code from the URL given earlier and print it out with line numbers.

All of rtp's non-automatic data structures are embedded within two primary C++ classes. **PlotWindow** is derived from Qt's QWidget and provides all user interface callbacks, as well as the STDIN callback. These classes are laid out in rtp.h. The important data members of PlotWindow are:

- **deque<DoubPt> _points**: the entire set of raw (x,y) data points received from stdin. **deque** is a C++ Standard Template Library class that (among other things) gives the illusion of contiguous memory layout (array indices, fake pointers that you can increment to move through the deque) to a dynamically sized block-linked data structure. Points are kept in the order in which they are received.
- **QPixmap *_buffer**: the pixmap that is copied into the plot window whenever the window is painted.
- **RtpMapping _map**: holds the viewport boundaries, scale factors and offsets currently in effect for mapping received data points into the plot window.
- **QRect *_rubberBox**: if non-NULL, defines the “rubber” box that the user is bounding with the mouse to define a new viewport. Once the user releases the mouse, the box will be deleted from the screen and the viewport will change.

The other important rtp class is **RtpRender**. Its important data members are:

- **QTimer _timer**: activated to call **RtpRender::slotWorkAwhile** while a rendering is in progress. Inactive when rendering is not in progress.
- **unsigned int _pti**: marks the position in points as successive calls to RtpRender::slotWorkAwhile progress through the data.
- **QPixMap *_privateBuff**: ***_buf. _privateBuff** will be created for a private rendering (explained below). **_buf** is the pixmap actually drawn into by RtpRender::slotWorkAwhile. It will equal either **_privateBuff** for a private rendering or the main repaint pixmap for an on-line rendering.

How rtp Processes Data Points

Data points to plot come in on STDIN. As part of its initial setup, rtp sets the STDIN file descriptor mode to non-blocking so that any read of STDIN will not block the program. This allows us to read stdin in relatively large chunks, increasing efficiency. **rtp** then creates a QSocketNotifier for STDIN, registering PlotWindow::slotStdinAwake as the callback through the signal/slot mechanism. Here is the code, from line 454 of rtp.c:

```
fcntl(STDIN_FILENO, F_SETFL, O_NONBLOCK);
QSocketNotifier sn(STDIN_FILENO,
  QSocketNotifier::Read);
QObject::connect(&sn, SIGNAL(activated(int)),
  plotter, SLOT(slotStdinAwake()));
```

When **slotStdinAwake** (rtp.c, line 100) is called, we know there is at least one character of data on STDIN (because select returned with STDIN marked as ready, and all POSIX I/O is character-based). However, simply reading one character and then returning is very inefficient. For optimal efficiency, we want to read and process as many characters as possible.

However, the time spent in slotStdinAwake contributes to the user interface latency, because no X events can be processed until slotStdinAwake exits. If we processed as many STDIN characters as were available and STDIN was receiving points at a faster rate than they could be processed, we could risk locking out the UI (user interface) completely. So we have a classic tradeoff between efficiency and latency. The current version of rtp attempts to read and process 1024 characters of data per slotStdinAwake call. However, because the **read** call doesn't block, we may not actually process this many characters.

slotStdinAwake is uglified by the fact that it does its own buffering and doesn't use the STDIO library. I couldn't tell from the GNU libc information whether STDIO would work after **O_NONBLOCK** had been set on the descriptor. Rather than find out, I took the lazy approach and wrote my own buffering code.

Listing 1

When rtp parses a new x,y data point, it will map it into the current pixmap if it is within the current viewport's range. If the point is out of range and the plot mode is auto-scale or auto-tracking, the entire plot must be redrawn at a new scaling and offset. The code in Listing 1 (rtp.c, line 255) handles these cases.

How Rtp Renders the Set of Data Points

Listing 2

The class **RtpRender** (defined in rtpRender.c) handles the details of drawing the set of data points into a pixmap. Because rendering the entire set can take significant time, RtpRender sets up a QTimer object with zero timeout to give all available CPU time to rendering while maintaining snappy UI response. **RtpRender::slotWorkAwhile**, the QTimer callback, munches on points for a fixed interval (100 msec at present), and then returns. The code in Listing 2 is the guts of RtpRender::slotWorkAwhile (rtpRender.c, line 274).

Listing 3

There are two types of rendering operations. A pre-emptive or on-line rendering draws points directly into the pixmap used for repaint events. A new on-line rendering is started by calling **RtpRender::newOnlineRender**. When this is called, any rendering that may be in progress is cancelled and the new rendering starts from scratch, drawing all received points. The code is in Listing 3 (rtpRender.c, line 77).

When RtpRender::newOnlineRender is called, a pointer to the pixmap used for repaint is passed in as the **buf** argument. **_map** is a data structure that contains the scale and offset factors for the new rendering and is returned to the calling code, so that new points from STDIN can be directly plotted even as the rendering progresses.

A non-preemptive or private rendering first creates a private pixmap, then draws the points into it. A private rendering is requested by calling **RtpRender::quePrivateRender**. The private rendering does not cancel the current rendering operation if one is in progress. It waits until the current rendering is complete before starting. The code is in Listing 4 (rtpRender.c, line 97).

Listing 4

Because the queuing mechanism is just a boolean flag, the private rendering queue is only one deep. When RtpRender::quePrivateRender is called, it will

destroy any pending private rendering operation, but will not interfere with one already in progress. Note that **_timer** is an object of type QTimer. If the timer is already active, it means a rendering is already in progress.

rtp uses private rendering to update the plot when new data points force a viewport change in either the auto-scale or auto-tracking mode. On-line rendering is used to update the plot in response to user-initiated viewport changes, such as zooming into a moused region. The theory is new points come in quickly enough that we don't want to start over each time we get one. However, when the user changes the viewport, he has no interest in anything but the latest and greatest plot.

Resources

David Watt can be reached via e-mail at wattd@elcsci.com.

Archive Index Issue Table of Contents

Advanced search

Shell Functions and Path Variables, Part 3

Stephen Collyer

Issue #73, May 2000

A continuation of our introduction to path variables and elements.

In this final article in the series, I'll describe the remaining path handling functions and point out a few implementation issues. Before I do that, however, I will describe a utility called **makepath**. This reads either standard input or its argument list, builds a colon-separated path variable (pathvar) from those lines read and echoes it to standard output. For example:

```
$ makepath /bin /usr/bin /opt/kde/bin
/bin:/usr/bin:/opt/kde/bin
```

makepath is used in several of the pathvar utilities to reconstruct a pathvar after its path elements (pathels) have been altered. I won't show you the innards of makepath, as they're somewhat tangential to the main topic and rather trivial.

listpath

First, let's look at **listpath**, which echoes the pathels making up a pathvar on separate lines, as in:

```
$ listpath -p MANPATH
/usr/man
/usr/local/man
/opt/CC/man
```

Using listpath has two advantages over merely echoing **$MANPATH**. First, it's much easier to read the pathels when they appear on separate lines; and secondly, you can pipe its output through **grep**:

```
$ listpath | grep bin
/opt/kde/bin
/usr/local/bin
/bin
```

There is no option-handling code we did not see in the **addpath** function, so let's look at the main code:

```
eval echo
```

This is very simple. We just echo the contents of the specified pathvar into the **colon2line** function (included in the tar file mentioned at the end of this article), which turns the embedded : characters into newline characters. I described the operation of this piece of code in some detail in Part 2, so I won't repeat it here. Take a look at that article (http://www.linuxjournal.com/lj.issues/issue72/3768.html) if you're not sure why the **eval** is there.

delpath

We have already seen the addpath function, which performs an idempotent addition of a pathel to a pathvar. The converse of this behaviour is provided by **delpath**, which removes pathels from a pathvar. So, for example:

```
delpath /opt/CC/test/bin
```

will remove the /opt/CC/test/bin directory from **$PATH** and:

```
delpath -e "(bill|steve)" -p MANPATH
```

will remove all pathels matching the **egrep**-style regular expression **"(bill|steve)"** from **$MANPATH**. The command

```
delpath -n
```

removes all non-existent directories from **$PATH**.

Although delpath is not a function you are likely to need often, there is one place where it can be useful. Many UNIX machines have a file called /etc/PATH, which is sourced by /etc/profile. It sets up a default PATH containing directories required by all users. Too often, though, /etc/PATH is not modified for years at a time, and the directories added either no longer exist or are not truly required by all. In this case, you can call delpath at the start of an appropriate login script (.profile or .bash_profile) to remove the directories you do not need.

Let's look at the delpath code. I'll skip most of the option handling, as much of it is identical to that in addpath.

```
MATCH="-x"            # default
[ -n "$opt_e" ] && MATCH= # make grep use regexps
FILTER=               # default
[ -n "$opt_n" ] && FILTER="| realpath_filter"
```

Here, we see the final section of the option handling. The **MATCH** variable determines whether we handle the supplied path description as a regular

expression or not. It is used later as an option to grep; **grep -x** tells grep to perform exact string matches.

The **FILTER** variable implements the **-n** option, i.e., the “remove non-existent directories” behaviour. If the user supplies the **-n** option, **FILTER** contains a string which pipes the output of a previous command through a program called **realpath_filter**. This program reads directory names from its standard input and writes the name to standard output only if it is an existing directory. I'll leave it as an easy exercise for the reader to implement such a filter.

The remainder of delpath is as follows:

```
eval listpath -p $pathvar $FILTER |
  grep -v -E $MATCH "$1"> /tmp/makepath_in.$$
eval $pathvar=$(makepath < /tmp/makepath_in.$$)
rm /tmp/makepath_in.$$
```

The function does its work in three stages. The first command generates a file in /tmp containing those directories that are not to be deleted. The second command rebuilds the pathvar from that file using makepath. Finally we remove the file; we don't want it cluttering up the file system after the function finishes. (The shell expands **$$** to the process ID of the shell running the command; I'll assume it is 20610 in this article.)

Let's look at the first line. Essentially, it uses **listpath** to break the appropriate pathvar into separate lines, and grep to remove those we don't want. It's slightly complicated, however, by the presence of the **FILTER** variable. Suppose the user types:

```
delpath -e "^opt"
```

which means “remove all directories starting with the **opt** string from **$PATH**”. In this case, pathvar will contain **PATH**, while **MATCH** and **FILTER** will be empty. The first line will therefore expand to:

```
eval listpath -p PATH |
  grep -v -E "^opt<" > /tmp/makepath_in.20610
```

This is straightforward—listpath writes the pathels in **PATH** into the grep command, which we use to echo non-matching lines only (**-v**). We redirect the output into our temporary file, which will contain those pathels not starting with **opt**. In this case, the leading eval is unnecessary. However, if the user types

```
delpath -n
```

to remove all non-existent directories from **$PATH**, then the first line expands to

```
eval listpath -p PATH | realpath_filter |
grep -v -E "" > /tmp/makepath_in.20610
```

During the initial processing of the line (i.e., before the eval has forced re-evaluation), the shell saw the pipe symbol preceding the grep, but it did not see the pipe symbol preceding realpath_filter. As things stand at the moment, the shell sees the first | as a literal character and will pass it as an argument to listpath. This happened because the shell looks for | characters before it expands variables, and the | character preceding realpath_filter was stored in a variable. The second evaluation caused by the eval ensures the pipeline that runs the realpath_filter command is constructed.

We now have a file containing only the required pathels. The second line in delpath rebuilds the pathvar from that file, using the following code:

```
eval $pathvar=$(makepath < /tmp/makepath_in.$$)
```

This shouldn't cause us too many problems. First, makepath simply reads the lines in the file, builds a colon-separated pathvar and echoes it. We run makepath in command-substitution mode (that's the **$(...)** which I described in Part 2), so makepath's output is used as the right-hand side of the variable assignment. The initial eval is required due to the order in which the shell evaluates a command. Because it looks for assignment statements before expanding variables, it won't recognize that the command contains a valid assignment. The eval ensures the assignment takes place the second time the line is processed.

uniqpath

Suppose you log on to your UNIX system and discover, for reasons beyond your control, that PATH is full of duplicate entries. (Humour me. It does happen. Maybe your system administrator modified /etc/PATH inadvisedly). Let's assume these duplicates are making your PATH undesirably long. Is there anything you can do to clean things up? Yes, you can type at the prompt:

```
$ uniqpath
```

This will remove any duplicate entries from your path, leaving the order of the remaining pathels intact. For example:

```
$ NEWP=fred:bill:steve:fred:dave:bill
$ uniqpath -p NEWP
$ echo $NEWP
fred:bill:steve:dave
```

Let's skip the options-handling code again, and look at the meat:

```
npath=$(listpath -p $pathvar | awk '{seen[$0]++;
if (seen[$0]==1){print}}')
    eval $pathvar=$(makepath "$npath")
```

As usual, **$pathvar** contains the name of the pathvar we want to modify. The code is rather similar to that of delpath. The first line generates a variable (**npath**) containing the unique path elements, and the second line rebuilds the pathvar from those elements using makepath. We don't use an external file to store the pathels, but keep everything in shell variables. This is done in order to demonstrate an alternative technique—there is no deeper reason.

The first line runs listpath to break the pathvar into separate lines and pipes them through an **awk** filter which removes duplicate pathels. You may be wondering why we don't just use the **uniq** program instead of awk's magic. It's because uniq will remove duplicate lines from its input only if they happen to be adjacent. In our case, the duplicate pathels will generally not be adjacent, so **uniq** won't work. “Aha,” you say, “why not use **sort -u**? That will sort the lines and remove duplicates.” True enough, however, it may also modify the directory search order, if we ran **uniqpath** to alter **PATH**. Usually, people care about the order in which their PATH directories are searched, and it's a bad idea to modify it.

Thus, we have the awk solution. This uses a powerful feature of awk known as an associative array or hash (if you have a Perl background). If you're a C programmer, you'll know what an array is: a group of objects of the same type, indexed by an integer. The contents of an array can be accessed by expressions like values[0] or values[20], which refer to the first and twenty-first elements, respectively. A hash is rather like an array which can be indexed by an arbitrary string of characters. So, in awk notation, we could write

```
age["bill"]=27
```

to assign 27 to the hash element indexed by the string **bill** in the hash called **age**. Let's look at the awk code shown above.

Between the single quotes, we have a block of code run each time awk reads a new line from its standard input. When awk reads a line, it is stored in a special variable called **$0**, and we use **$0** as an index into a hash called **seen**. (We haven't declared this anywhere—that's okay in awk. Variables spring into existence, with numerical value 0, when they appear in the code). We use the **seen** hash to tell us whether awk has already seen an identical line of input since it started executing. Let's see what happens in the **NEWP** example shown above.

First, listpath splits **NEWP** into lines containing the following strings: “fred”, “bill”, “steve”, “fred”, “dave” and “bill”, which are read in that order by awk. **awk** stores each line it reads in **$0**, so **$0** takes on the values “fred”, “bill” and so on, in turn. Each time a line is read, the corresponding element of the **seen** hash is incremented (by the line **seen[$0]++**) and is printed only if it has been seen

exactly once (by the print statement in the **if** block, which prints **$0** to standard output by default). If we look at the hash element **seen["fred"]**, this is initially 0 and is then set to 1 when awk reads the first “fred” line, remains at 1 for the next two lines, and is set to 2 when awk reads the second “fred” line. It is printed only when it is seen for the first time. C programmers should note how syntactically elegant this solution is and how little code is required when compared to the equivalent in C.

edpath

The final pathvar function we're going to see is **edpath**. This breaks the pathels in a pathvar into separate lines, writes them to a temporary file and runs an editor on that file. You can edit the pathels to your heart's content and quit from the editor when you're finished. The pathvar is then reconstructed from the modified lines in the file. **edpath** allows you to perform arbitrary modifications on a pathvar. I use it most often when I wish to swap the order of directories in **PATH**.

The code for edpath is fairly straightforward (ignoring once again the boring details of option handling):

```
TEMP=/tmp/edpath.out.$$
VAR=\$$pathvar  # VAR="$LIBPATH" for example
eval export OLD$pathvar=$VAR  # store old path in
                              # e.g. OLDPATH
listpath -p $pathvar > $TEMP  # write path
                              #  elements to file
${EDITOR:-vi} $TEMP           # edit the file eval
$pathvar=$(makepath < $TEMP) # reconstruct path
/bin/rm -f $TEMP           # remove temporary file
```

Let's skip the first three lines for now. The real work is done by the block of code starting with listpath. This follows a similar pattern as delpath and uniqpath. First, we separate the pathels in the pathvar using listpath, but this time, we redirect the output into a temporary file. The next line edits that file. The expression **${EDITOR:-vi}** may be unfamiliar; it means “Use the value of the **EDITOR** variable if it is non-null, else use vi.” This allows the user to specify his favourite editor by setting the **EDITOR** environment variable (to Emacs, perhaps) but uses vi if he has not done so. Note that the **edit** command is run in the foreground, so the shell will wait until the editor process terminates before running any more commands from the shell function. When this occurs, the modified pathvar will be reconstructed by the line starting with **eval**. If you read the description of delpath given above, you'll know how this line works.

Lines 2 and 3 of the code are a safety net. They store the initial value of the pathvar to be edited in a new environment variable. If the user is editing **PATH**, for example, then the code creates a variable called **OLDPATH**. If the user makes unwanted modifications to her **PATH**, she can simply type:

```
$ PATH=$OLDPATH
```

and all will be well.

Conclusion

UNIX can present a bewildering array of tools and techniques, and it's almost impossible for any individual to be intimately familiar with all of them. In my experience, the best developers carry around a large bag of simple but useful techniques and are able to combine them rapidly into a working solution. You don't need to know every detail of every tool to do useful work, but you do need a bag of tricks you understand.

Please feel free to use any of the ideas I've described in this series. You can get a hold of the source code to the shell functions from www.netspinner.co.uk/Downloads/pathfunc.tgz. Let me know if you find any bugs, would like a new feature added, or make an improvement.

Stephen Collyer (stephen@twocats.demon.co.uk) is a freelance software developer working in the UK. His interests include scripting languages and distributed and thread-based systems. Occasionally, he finds the time to talk to his wife and two remarkably attractive and highly intelligent children.

Archive Index Issue Table of Contents

Advanced search

IBM's Universal Database

Paul Zikopoulos

Issue #73, May 2000

Getting DB2 up and running on Linux.

Have you been trying to install DB2 Universal Database on a Linux-based workstation? Did you run into some troubles? As I scan around the DB2 and Linux newsgroups, I hear from many of users who are getting frustrated when trying to get DB2 running on Linux. How did this problem get so big? Well, the Linux phenomenon is relatively new and ever-changing. Recently, all sorts of vendors are flocking to market their distributions, with slight differences between them all. Combine that with what seems to be quarterly releases and you can see how communication channels between the Linux vendors and the people who build applications to run on them get clogged. While efforts are being made between application developers and Linux vendors to define this communication pipe, you can use the information in this article to get yourself up and running in no time at all.

DB2 for Linux is officially supported on the following Linux distributions: Caldera OpenLinux, Red Hat Linux, TurboLinux and SuSE Linux. This article will take you through the steps involved in installing DB2 on each of the supported Linux distributions. In the article, I assume you have not previously installed a version of DB2 and you are not maintaining any of the default users created by a default DB2 Installation. The three user IDs that will be created during a DB2 installation are: db2inst1, db2fenc1 and db2as. If you have any of these users on your system, be sure to remove them and their associated directories before installing DB2. This article also assumes you are familiar with the **rpm** command, used to install packages. If you are not familiar with this command, refer to your Linux documentation.

In order to run DB2 on Linux, the following are required:

- Linux kernel 2.0.35 or higher
- RPM (package manager)

- pdksh package (public domain Korn shell)
- glibc version 2.0.7 or higher
- libstdc++ version 2.8.0

Preparing Your Linux Workstation for a DB2 Installation

The following sections will highlight any actions you need to perform on your Linux workstation to enable it for a DB2 installation. If a requirement exists by default on your system, I will not make note of it in the sections that follow. Keep in mind that the lag time between when this article was written and when you are going to read it may be a couple of months. In that time, some of the levels or links I have referred to here may have changed.

DB2 and Caldera OpenLinux

The information I am providing in this section is based on a Caldera OpenLinux version 2.3, or simply Caldera, Standard Installation type. If you installed a different installation type on your Linux workstation, you may have to add some of the required packages to your workstation.

The pdksh package is missing from the default Standard Installation. This package is available on the Caldera CD-ROM; however, it is not compatible with DB2. IBM and Caldera are working to solve this problem, but in the meantime, I recommend you download a pdksh package from a Red Hat mirror site—it will work just fine. I am sure Caldera Systems will post a fix sometime on their FTP site at ftp://ftp.calderasystems.com/pub/.

For now, go to the mirror site at http://www.metalab.unc.edu/pub/Linux/distributions/redhat/redhat-6.1/i386/RedHat/RPMS/. Download the pdksh-5.2.14-1.i386.rpm package and install it with the rpm command using the **-nodeps** option. If you try to install this package without the **-nodeps** option, you will receive an error stating that this package has a requirement on the glibc package. This error is only a result of the different naming conventions used by Caldera Systems and Red Hat. A glibc package is installed by default.

For DB2 version 6, you require libstdc++ 2.80, DB2 v6 will not run with libstdc++ 2.9.0. Caldera version 2.3 by default installs with libstdc++ 2.9.0. The required libstdc++ 2.8.0 is located on the CD-ROM in the /col/contrib/RPMS directory as a package called libstdc++-compat-2.8.0-1.i386.rpm.

If you are running Caldera version 2.2, I recommend that you upgrade to version 2.3; it will make your DB2 installation much easier. In the event you do not want to do this, there are some things you will need to note on top of the issues I mentioned for Caldera version 2.3.

First, the libstdc++-compat-2.8.0-1.i386.rpm package, is not on the CD-ROM. You can get it from Caldera at ftp://ftp.calderasystems.com/pub/openlinux/2.3/contrib/RPMS/.

Finally, DB2 requires a file called libcrypt.so.1 to work. This file is usually shipped with every Linux distribution. Some problems with federal export laws caused Caldera version 2.2 to ship without this file. To add this file to your workstation, download the package glibc-crypt-2.1-3i.i386.rpm from ftp://ftp.linuxland.de/pub/OpenLinux/crypto/2.2/RPMS/. I could not find this fix on Caldera's FTP site.

Once you have completed these tasks, your Caldera version 2.2 workstation is ready for a DB2 installation.

DB2 and TurboLinux

The information I provide in this section is based on a TurboLinux version 3.6 Base Workstation installation. If you installed a different installation type on your Linux workstation, you may have to add some of the required packages to your workstation.

There are some problems trying to get DB2 to run on a workstation running TurboLinux. Download a fix from the Web at: ftp://ftp.software.ibm.com/ps/products/db2/tools/. The fix is called tl36_instfix.tar.Z, note that the l is the letter “l” not the number “1” All the information you require to implement this fix is mentioned in the README file called tl36_instfix.readme.txt.

After you have downloaded the fix, you need to add the pdksh package, which is not part of the Base Workstation installation. This file is available on the TurboLinux CD-ROM, in the /TurboLinux/RPMS directory.

Once you have completed these tasks, your TurboLinux version 3.62 workstation is ready for a DB2 installation.

DB2 for SuSE Linux

The information that I am providing in this section is based on a SuSE version 6.3 Network Oriented System installation. These instructions also apply to a workstation running SuSE version 6.2. If you installed a different installation type on your Linux workstation, you may have to add some of the required packages to your workstation.

The biggest problem with installing DB2 on a workstation that is running SuSE Linux is the naming convention that SuSE uses for its packages. For example, SuSE calls the required glibc package shlibs. This will causes problems when

you try to install DB2 because the DB2 installation utility will fail to recognize the existence of the required glibc package. To get around this problem, you have to install a dummy package, called glibc-2.0.7-0.i386.rpm. This package is located in the /db2/install/dummyrpm directory on your DB2 product CD-ROM.

Additional Steps for SuSE Version 6.1

SuSE Linux version 6.1 ships with a beta copy of the DB2 for Linux version 5.2 code. Consequently, when you go to install DB2, this causes problems with the default users. To make things ever stranger, I noticed that when I installed the Network Oriented System installation, which was not supposed to include DB2, the default DB2 users were created. To make matters worse, I could not find any information about the passwords for the DB2 users that SuSE creates (they are not the default DB2 passwords), and some of the settings that SuSE implements do not work for DB2. In the end, remove the users (db2inst1, db2as, db2fenc1) that the SuSE installation creates. For more information on SuSE user management, refer to your product's documentation.

Once you have completed these tasks, your SuSE version 6.1 workstation is ready for a DB2 installation.

DB2 and Red Hat Linux

The information I provide in this section is based on a Red Hat version 6.0 Server installation. These instructions also apply to a workstation that is running Red Hat version 5.2, though the names of the packages may be at a different level. If you installed a different installation type on your Linux workstation, you may have to add some of the required packages to your workstation.

Both the Red Hat version 5.2 and version 6.0 installation are easy to enable for a DB2 installation. They are both missing the required pdksh package that is required to run the DB2 Installer. This package is located in the /RedHat/RPMS directory on the Red Hat CD-ROM.

If you are trying to install DB2 on a workstation that is running Red Hat version 6.1, you aren't going to get very far due to a problem with this version of Red Hat v6.1 and DB2. You can download the Red Hat fix at ftp.software.ibm.com/ps/products/db2/tools. The fix you need depends on where you got your DB2 code. If you are installing the copy of DB2 bundled with Red Hat 6.1, download the file db2rh61fix.tgz. If you are installing any other DB2 code, you need to download the db2rh61gafix.tgz file.

After you download the appropriate fix, unpack them by entering the **tar xvzf *filename*** command, where *filename* is the name of the downloaded fix file.

After unpacking this file, you will see three files in the directory. Once of them is a README file, called readme.txt. This file gives complete and detailed instructions on how to implement this fix.

Once you have completed these tasks, your Red Hat version 6.1 workstation is ready for a DB2 installation.

Installing DB2

Installing DB2 is made easy by an installation and setup utility called DB2 Installer. This utility will install all required packages for DB2, create instances for DB2 databases and administration support, and configure your DB2 server for communications. The instructions in this section assume you do not change any of the defaults presented by DB2 Installer, unless noted in the steps below.

You will usually run into display problems if you start DB2 Installer from a command window in your Linux distribution's graphical interface. You can refresh the view of DB2 Installer at any time by pressing **CTRL**-L. To avoid most potential display problems, I recommend running the DB2 Installer in a virtual console session outside of your operating system's graphical interface. You can shift between the virtual console session and the graphical interface session on most Linux distributions by pressing **CTRL**-**ALT**-**F1** and **CTRL**-**ALT**-**F7**. Refer to your Linux documentation for more details.

To quickly install DB2, perform the following steps:

- Log on to the system as a user with root authority
- Mount the DB2 product CD-ROM by entering the following command:

```
mount -t iso9660 -o ro /dev/cdrom /mnt/cdrom
```

- where /mnt/cdrom is the mount point of the CD-ROM. Note that even if you are installing DB2 on a workstation that is running TurboLinux version 3.6 or Red Hat Linux version 6.1, you still need to mount the CD-ROM. The image created by the install fix links to the DB2 CD-ROM. For more information, refer to your fix's readme.txt file.
- Change focus to the mount point of the CD-ROM. Note that if you were installing DB2 on a workstation that is running TurboLinux version 3.6 or Red Hat Linux version 6.1, you would change to the directory where you created the image on your workstation. For more information, refer to your fix's readme.txt file.

- Enter the **./db2setup** command to start the DB2 Installer program. The Install DB2 V6.1 window will open. The contents of this window vary with

respect to the DB2 product you are installing. Figure 1 is the window displayed when you are installing DB2 Workgroup Edition for Linux.

- tab key to move the selector bar, and the **ENTER** key to select or deselect an option. For more information or assistance during the installation of DB2, select Help. From the product list, select the DB2 product you want to install. For this example, select DB2 UDB Workgroup Edition, then OK. The Create DB2 Services window will open.
- Select the “Customize” option beside the DB2 product you want to install. For our example, select the “Customize” option beside the DB2 UDB Workgroup Edition option.
- Select the “Java Support” and the “Control Center” components.
- Select the “Create a DB2 Instance” option. Enter a password for this user and verify it by retyping this password in the field provided.
- Select the “Properties” option.
- Select the “Create a Sample Database” for a DB2 instance then OK.
- Select OK.
- A window will open that asks you to create a user that will be used to execute user-defined functions (UDFs) and stored procedures. For this example, you do not need to know anything about, or use, this user. Simply enter a password for this user, verify it and select OK. You are returned to the DB2 Create Services window. For more information on UDFs or stored procedures, refer to the “Administration Guide”.
- Select the “Create the Administration Server” option. Enter a password for this user as well and select OK.
- A pop-up window will open telling you the DB2 system's name for this workstation. Select OK. You are again returned to the Create DB2 Services window.
- Select OK, then Continue, and finally, OK to begin the installation. When the installation completes, respond to the DB2 Installer's prompts to close this utility.

You are finished all the steps necessary to install DB2!

Verifying your DB2 Installation

Now that you have finished your installation, let's go through a quick sample query to prove to you that this powerful database is actually alive and running on your system.

To verify your installation, perform the following steps:

- Log on to the system as the db2inst1 user. If you use the **su** command to do this, ensure that you enter this command with the **- option**; for example, **su - db2inst1**.
- Enter the following command to connect to the sample database created by DB2 Installer:

```
db2 connect to sample
```

- You should receive output that is similar to the following:

```
Database Connection Information
Database server = DB2/LINUX 6.1.0
SQL authorization ID = DB2INST1
Local database alias = SAMPLE
```

- Enter the following command to select a list of all employees who belong to department 20 in the staff table:

```
db2 "select * from staff where dept = 20"
```

- Note that you must enter this command using the quotation marks so that your operating system does not confuse the SQL statement with a command. You should receive output that is similar to that shown in Table 1.
- End the database connection by entering the following command: **db2 terminate**.

Table 1

Enabling the DB2 Graphical Tools

You must have the prerequisite Java Runtime Environment (JRE) level to use the DB2 Administration Tools. For more information, refer to the Control Center README, which can be found in the INSTHOME/sqllib/cc/prime directory, where INSTHOME is the home directory of the user created for the instance during the installation (for example, /home/db2inst1/sqllib/cc/prime). Let me save you some time and take you through the quickest way I found to get the graphical tools running on your workstation. Perform the following steps:

- Log on to your workstation as a user with root authority
- In order to run the graphical tools, you need to be running at least Java 1.1.7 v3 or later. A JRE is available from the Blackdown web site at www.blackdown.org/java-linux/mirrors.html. For this example, I visited the mirror site at: ftp://metalab.unc.edu/pub/linux/devel/lang/java/blackdown.org/JDK-1.1.7/i386/glibc/v3/, downloaded the jre_1.1.7-v3-glibc-x86-native.tar.gz file and placed it in the /tmp directory. For all the latest

information on supported JREs and browsers, go to http://www.software.ibm.com/data/db2/.

- Note that you must download the native threads version of the JRE that you want to use. The DB2 Control Center does not support green threads.
- Once you have downloaded an appropriate JRE, unpack the file by entering the following command:
- tar xvfz jre_1.1.7-v3-glibc-x86-native.tar.gz
- Log on to your workstation as the db2inst1 user.
- Update your PATH so that your workstation knows the location of the JRE's binary files just installed. Assuming you are following along the example, for Bash or Bourne shells enter this command: **export PATH=/tmp/jre117_v3/bin:$PATH**; for C shell enter: **setenv PATH /tmp/jre117_v7/bin:${PATH}**, where /tmp/jre117_v7/bin is the path to the downloaded JRE binary files.
- Start the JDBC Applet Server by entering the following command: **db2jstrt 6790**.
- Start a graphical administration tool. For this example, let's start the Control Center using the command **db2cc 6790**.
- The Control Center Sign On window opens. Enter a valid DB2 user that has SYSADM authority on the instance with which you intend to work. For our example, enter the db2inst user ID and the corresponding password. For more information, refer to the “Administration Guide”.

Now, you have completed all the steps necessary to configure your workstation for the DB2 graphical tools.

If you followed the steps and considerations that I have outlined in this article, you should have a running copy of DB2 on your Linux workstation, a sample database, a running Control Center and a smile on your face!

Paul Zikopoulos (paulz_ibm@yahoo.com) is a senior member of the DB2 information and development team, specializing in DB2 installation and configuration issues. He has more than five years experience with the DB2 Universal Database, and has written magazine articles and books about DB2. Paul is an IBM Certified Advanced Technical Expert (DRDA and Cluster/EEE) and an IBM Certified Solutions Expert with DB2 Universal Database.