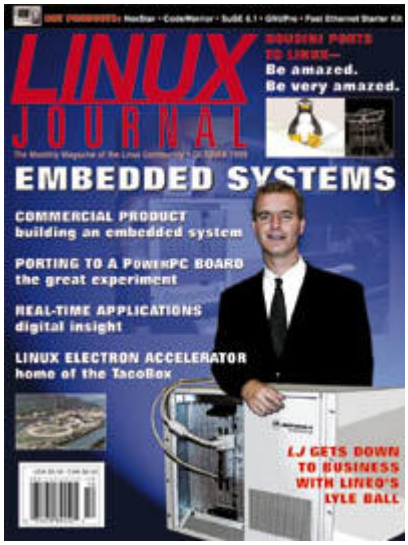
Linux Journal Issue #66/October 1999



Reviews

Columns

Departments

Strictly On-Line

Mr. Curley takes a look at two open-source solutions for embedded systems: RTLinux and eCOS from Cygnus.

Archive Index

Advanced search

Focus: Embedded Systems

Marjorie Richardson

Issue #66, October 1999

This month, we take a good look at how embedded systems work and how to build one that is cross-platform-compatible.

Today, embedded systems can be found wherever you look, from elevators to power company facilities to electron accelerators. They form a very important, though mostly unseen, part of our lives. This month, we take a good look at how embedded systems work and how to build one that is cross-platform-compatible. Embedded systems require real-time programming, and we have an article discussing this topic. Is there truly such a thing as real time? Read it to find out.

Linux's stability and performance make it a perfect choice for use as an embedded system, but size is another matter. Downsizing can be a difficult task. Just making the decisions regarding what to cut and what to keep can be both time-consuming and complex. Is it worth it?

One company has bet their business that it is. Caldera Thin Clients has changed its name to Lineo, Inc. and is coming out with an embedded Linux system. Called Embedix, this system is based on Caldera's OpenLinux. I talked to Lineo's Lyle Ball to find out about this product and future plans for the company. (The interview will be in our next issue.)

Another company chose a different road. Cygnus wrote their own embedded operating system, called eCOS, which they have released as open source. Charles Curley has written an article comparing eCOS to RTLinux, which appears in the “Strictly On-Line” section on our web site, http://www.linuxjournal.com/.

Elsewhere in this issue, Craig Knudsen tells us why Transvirtual made the decision to adopt Microsoft Java Extensions, and Michael Hammel talks about

Side Effects Software, their port of Houdini to Linux and what it means to the community.

At LinuxWorld on August 11, I had the pleasure of having lunch with our leader, Linus Torvalds. As a change of pace for both of us, I took the opportunity to talk to him about personal matters rather than Linux. (See interview in issue 67.)

—Marjorie Richardson

Archive Index Issue Table of Contents

Advanced search

Portable Real-Time Applications

Juergen Kahrs

Issue #66, October 1999

“The purpose of computing is insight, not numbers.” --Richard Hamming, 1915-1998

In today's visual world of data processing, many people think solving problems with computers means implementing graphical user interfaces (GUIs). From this point of view, writing real-time applications means writing GUIs while at the same time mastering the system-dependent functions that ensure predictable response times, often in conjunction with arcane hardware features. This mixture of system-dependent GUIs and hardware-dependent real-time functions usually leads to complex, expensive and non-portable applications. To attack the general problem of writing portable real-time applications, I will first look back at the roots of the UNIX operating system. Then, I will apply the lessons learned to a simple multimedia application that runs on three very different platforms: Linux, IRIX and Win32 (MS Windows 95/98/NT/2000).

Good Vibrations

Most people know how a sine or a square wave sounds. They can be heard as a beep or a test signal from a PC speaker, a telephone or a common musical instrument such as a flute. The portable application we will implement produces such sounds. Sine, triangle and square waves are just special cases of a more general wave produced by the Duffing oscillator (see Resources). Depending on the parameters, which can be adjusted by two sliders in the GUI, this oscillator is also able to emit chaotic sounds. (Chaotic, in this sense, gets its meaning from nonlinear dynamics or chaos theory.) When starting the application, you will actually be able to put some research results to the test by listening to the sound and by watching the graphical behaviour of the Duffing oscillator. Nonetheless, such an application has to produce sound continuously, otherwise the sound will be distorted by clicks or even silence. Thus, this application is an example of a real-time application.

The UNIX Philosophy

Before starting our implementation, we must think about the design of such a portable application, forgetting for the moment that we want to implement our multimedia application with a GUI. Most UNIX programmers know what is meant by the *UNIX philosophy*. In the good old days, when all user interfaces were textual, not graphical, Kernighan and Pike explained this notion in the epilogue of their book *The Unix Programming Environment*. They emphasized the importance of breaking a problem into separate sub-problems with a simple interface between them, usually a pipe. Putting a pipe between the processes means having the textual output of one process read as textual input by the next process. Each sub-problem was then implemented and tested stepwise on its own, preferably by applying existing tools.

This design philosophy allows for writing portable applications and is a sharp contrast to today's development environments. Today, many programmers use some visual development environment to build monolithic applications bound to one platform.

The Pipeline

The crucial question is: can a GUI-based real-time application be implemented as an old-fashioned UNIX pipeline? It can—you just have to choose the right tools. A GUI-based application which allows for adjustment of parameters, emission of sound and visualization of results can be broken down into a pipeline of processes: GUI-->Sound Generation-->Sound Output-->Graph.

Stage 1 (GUI) provides a way to interactively adjust the parameters of a mathematical model. These parameters may be adjusted with knobs, slide bars or with a point in a two-dimensional plane, whichever is most intuitive. It can easily be replaced without affecting other stages, provided it produces the same kind of data on its standard output.

Stage 2 (generation) is invisible to the user, so it does not need a GUI. It takes the parameters from stage 1 and processes them using a mathematical model of a physical process. The only challenge at this stage is doing both parameter input and continuous computation at the same time, but at different unsynchronized rates.

Stage 3 (sound emission) reads the resulting waveform and hands it over to the sound system. Because of significant differences in the implementation of sound systems on platforms like Linux, IRIX and Win32, we need to have some experience with encapsulating platform-dependent code. Fortunately, this is the only stage written for a particular platform.

Stage 4 (graphical output), just like stage 1, has contact with the user and will therefore be implemented as a GUI. Just like stage 1, the results can be represented in many different ways without affecting other stages. Examples are tables, simple amplitude diagrams, trajectories in phase space or Poincaré sections.

Each stage can also be used on its own or as a building block in a completely different application. Stage 3 is the most interesting building block.

Choice of Tools

It is clear that transferring large amounts of data from stage 2 to stage 3 is the bottleneck of the system. Writing into the pipe, reading again and scanning each sample absorbs more CPU time than every other operation. So, stages 2 and 3 must be integrated into one program, because passing of data (44,100 values per second) takes too much time. You might think integrating these stages into one is a design flaw—it is not. I could have easily renumbered stages and changed this article, but I preferred to show you how cruel life in real time is to ingenuous software designers.

For Stage 1, Tcl/Tk is a natural choice as a tool for implementation of the first sub-problem. Many people have forgotten that the GUI process in Tcl/Tk also has a textual standard output which can be piped into the second process.

In Stages 2 and 3, the sound generator reads the textual parameters from the GUI-process and computes the proper sound signal from it in real time. Therefore, it should be written in C, because it is the most critical sub-problem as far as real-time constraints are concerned. As a side effect of sound generation, the data needed for graphing the results will be written to standard output and piped into the final stage of the application.

Since Stage 4 outputs graphs of the results, it is also a task well-suited for a tool like Tcl/Tk.

The user will notice only stages 1 and 4 of the pipeline, because he can see each of them as a window and interact with them. It is an interesting paradox that the seemingly important stages 1 and 4 are rather trivial to implement, given a tool like Tcl/Tk. Stages 2 and 3, although mostly unnoticed, are the most challenging sub-problems because of synchronization concepts in real time (**select** or thread), real-time constraints due to continuous sound emission and the different handling of sound systems and all platform dependencies.

Concurrency of Development

We have already mentioned that one advantage of this pipeline approach is splitting the development into largely independent sub-tasks, which allows one programmer to work on each task concurrently. Equally important is the fact that each stage could also be implemented in different ways by different programmers. To demonstrate this, we will look at three solutions to the stage 1 task:

- Textual user interface on the command line
- GUI with Tcl/Tk (Listings 1 and 2)
- GUI with a Netscape browser and GNU AWK 3.1 as a server (Listing 3, which is not printed but is included in the archive file)

Also, we will look at three different implementations of stage 4:

- Textual output into a file
- Graphical output of this data with GNUPLOT (Figure 5)
- Graphical output with Tcl/Tk (Listing 5)

How Will It Sound?

Now that the design of the system is clear, it is time to become more precise about the type of sound we want to produce. Imagine a driven steel beam held pinned to fixed supports at the bottom and top. When driving the beam from the side, the fixed supports induce a membrane tension at finite deflections. This leads to a hardening nonlinear stiffness for moderately large deflections by a cubic term. At the beginning of this century, the engineer Georg Duffing from Berlin, Germany was annoyed by this kind of noise which came from vibrating machine parts. Such noise is not only a nuisance, it also shortens the expected lifetime of machine parts. Duffing found a simple nonlinear differential equation which describes the behaviour of machine parts under certain circumstances:

```
x'' + kx' + x3 = Bcos(t)
```

This oscillator is driven by a sinusoidal force on the right of the equation (with magnitude *B*) and damped by the parameter *k* on the left side of the equation. So, there are only two free parameters in this driven oscillator.

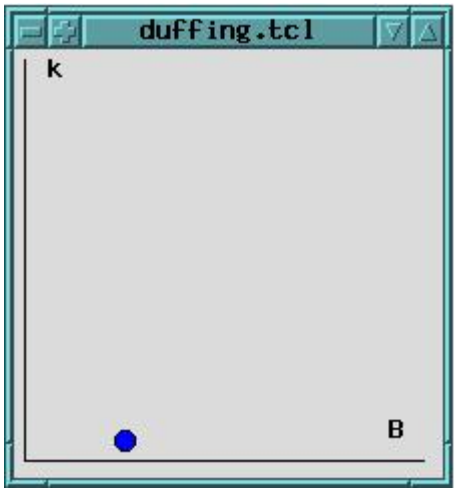


Figure 1. Changing parameters *k* (damping) and *B* (forcing) will move the oscillator in or out of a chaotic regime (see page 11, Thompson/Stewart).

In Figure 3, you can see a short wave form originating from such a sound machine. Unlike Duffing, you can simulate the noise production with your computer by varying the parameters with a GUI such as the one shown in Figures 1 and 2. You should expect that varying parameter *B* on the right axis of figure 1 influences just the volume of the noise. In fact, by pushing *B* to its minimal position 0, you can actually switch off the noise. When pushing parameter *B* to its maximum, noise will not only become louder, it will also change the main frequency but not in a continuous and monotonic way. This strange behaviour of changing frequency along with loudness does not occur in linear oscillators. In 1980, Ueda published a systematic look at the points in the plane opened up by the parameters *B* and *k* in Figure 1. By computer simulation, he found areas where the oscillator emits chaotic sounds. These results are summarized in Thompson and Stewart's book (see Resources).

Why is it so hard to compute these wave forms? After all, a formula to be evaluated for each time instant should be all that is needed; however, there is no such analytical function. When in trouble, engineers often fall back on simple approximations. We will do so with a technique called Finite Differencing (see Resources) which gives us a one-line calculation at each time instant (Listing 4).

The Implementation

Listing 1.

With Tcl/Tk, it is so simple to implement stage 1 that we can afford to look at two different implementations. When executing the script in Listing 2 with the **wish -f duffing.tcl** command, the two independent parameters *k* and *B* are visualized as the axes of a two-dimensional coordinate system. Drag the circle to any place on the map, and the new coordinates will be printed to standard

output. But maybe you prefer the simpler implementation shown in Listing 1, which displays both parameters as scales. If you run a computer without a GUI, you can still work with the software presented here. In this case, forget about stage 1; stage 2 will read its input from the command line. Enter lines like **k 0.05** or **B 7.5** at run time, and stage 2 will not notice the difference.

Listing 2.

Stages 2 and 3 had to be integrated into one program for reasons of efficiency. It was tempting to implement each stage as a single thread of execution. Threads of execution behave mostly like processes that share a single data space: one waiting for input to modify parameters, the other one calculating the wave form to be emitted. How does one implement threads of execution in a portable way? The POSIX thread library (see Resources) is now available for most operating systems, including the ones mentioned earlier. For example, at STN Atlas Elektronik we use threads for sound generation with multiple sound cards in a multiprocessor setting (two CPUs and Linux 2.2 SMP). As explained in David Butenhof's excellent book, threads make it very hard to debug software; therefore, we will refrain from using them here.

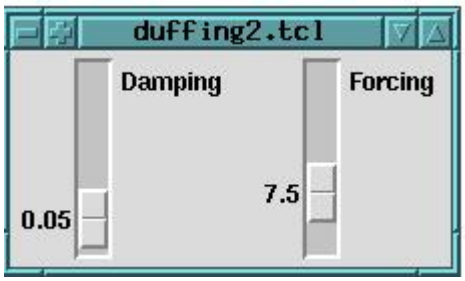


Figure 2. This user interface allows for precise tuning of parameters. Notice that you can actually hear chaotic regimes; they sound markedly “dirty” and not as “clean” as other regions.

Listing 4. Stage 2 and 3 in C, integrator.c

Fortunately, the problem of dealing with unsynchronized events can be solved with the often underestimated select system call (Listing 4, function **main**). The main loop of Listing 4 has a short loop to check whether there is data coming in from standard input. Then, it calculates a block of data by Finite Differencing and finally emits it. While calculating, some data points are printed to standard output. Only those which occur at integral multiples of the cycle period of the driving force, having the same phase angle, are printed. This technique of selecting data points to display is at the heart of the Poincaré section, a kind of stroboscope which reveals hidden order within chaotic data (Figure 5). Notice that this data is the input to stage 4.

Figure 3. The default parameters show sensitive dependence on initial conditions (see page 4, Thompson/Stewart).

Figure 3

This was the easy part; the hard part is handing over the data to the sound system in a portable way. In this respect, Linux is the platform handled most easily. Writing data to the special file /dev/dsp is enough. With IRIX, we also need just one function call; not the usual **write**, but a special sound function. In both cases, synchronization is implemented by the blocking behaviour of these functions. This is in contrast to Win32, which bothers the programmer with buffer handling, and synchronization must be done with callback functions. Using the new DirectSound API was not an option because Microsoft has failed to implement the DirectX API in Windows NT.

How to Compile and Run Listing 4

Listing 5. Stage 4 in Tcl/Tk, out.tcl

Stage 4 is, again, rather simple to implement (Listing 5, Figure 5). Each data point read is printed as a dot in the phase space diagram. When producing Poincaré section data as in stage 3, linear oscillators produce circles or spirals, degenerating into fixed points, which is rather boring. A chaotic oscillator is needed to plot the strange attractor shown in Figure 5.

Figure 4. High resolution Poincaré section of one chaotic attractor

Figure 4.

Portability Considerations

In the sidebar “How to Compile and Run Listing 4”, you can see how to compile the stage 3 program on different platforms and how to start the whole application as a UNIX pipe. You might be surprised to find that the compiler **gcc** can be used even with the Win32 operating systems. How is this possible? The gcc for Win32 is part of a UNIX-compatible environment called the Cygwin Toolset (see Resources). It allows you to work with gcc and its friends on any Win32 operating system just as if it were a well-behaved UNIX system. Many GNU packages run out of the box with it. If you want to use the multimedia functions of Win32 (along with DirectX access) or the POSIX thread library, you must get and install them separately (see Resources). Since I started working on Win32, it became more and more important to adhere to the POSIX system calls, written down in the XPG4 set of standards. Doing so has become a rewarding habit, because it is the cornerstone of portability in the UNIX world.

What pleased me most was the possibility of installing gcc from the Cygwin Toolset as a cross compiler on my Linux 2.2 system. Now I can write and debug my software with Linux, no matter where it is supposed to run later. In the end, I compile it with the cross compiler and get a debugged Win32 executable. What an achievement! If you also would enjoy working this way, follow Mumit Khan's recipe for cooking a cross compiler (see Resources).

Resources

Jürgen Kahrs (Juergen.Kahrs@t-online.de) is a development engineer at STN Atlas Elektronik in Bremen, Germany. There, he uses Linux for generating sound in educational simulators. He likes old-fashioned tools such as GNU AWK and Tcl/Tk. He also did the initial work for integrating TCP/IP support into gawk.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Embedding Linux in a Commercial Product

Joel R. Williams

Issue #66, October 1999

A look at embedded systems and what it takes to build one.

Most Linux systems run on PC platforms; however, Linux can also be a reliable workhorse for embedded systems. This article gives an overview of embedded systems and demonstrates what is involved in using Linux in a commercial embedded system.

Embedded Systems—Older than Moses

The computers used to control equipment, or embedded systems, have been around for almost as long as computers themselves.

In communications, they were used back in the late 1960s to control electro-mechanical telephone switches and were called “Stored Program Control” systems. The word “computer” was not as ubiquitous back then, and the stored program referred to the memory that held the program and routing information. Storing this logic, instead of hard-wiring it into the hardware, was a real breakthrough concept. Today, we take it for granted that this is the way things work.

These computers were custom-designed for each application. By today's standards, they look like a collection of mutant deviants, with strange special-purpose instructions and I/O devices that were integrated with the main computing engine.

The microprocessor changed that by providing a small, low-cost, CPU engine that could be used as a building block in a larger system. It imposed a rigid hardware architecture based on peripherals connected by a bus and provided a general purpose programming model, which simplified programming.

Software also advanced along with the hardware. Initially, only simple program development tools were available for creating and testing software. The runtime software for each project was usually written entirely from scratch. This was almost always written in assembly language or macro languages, because compilers were often buggy and lacked decent debuggers. The idea of software building blocks and standardized libraries did not come into vogue until the mid 1970s.

Off-the-shelf operating systems for embedded systems began to appear in the late 1970s. Many of these were written in assembly language, and could be used only on the microprocessor for which they were written. When microprocessor became obsolete, so did its operating system, unless it was rewritten to run on a newer microprocessor. Today, many of these early systems are just a faint memory; does anyone remember MTOS? When the C language came along, operating systems could be written in an efficient, stable and portable manner. This had instant appeal to management, because it held the hope of preserving the software investment when the current microprocessor became obsolete. This sounded like a good story in a marketing pitch. Operating systems written in C became the norm and remain so today. In general, reusability of software has taken hold and is doing rather nicely.

My favorite OS in the early 1980s was the Wendon operating system. For about $150, you received a library of C source code. It was a kit, and you built your own operating system by choosing components—kind of like ordering dinner from a Chinese menu. For example, you could pick a task scheduler algorithm and a memory management scheme from a list of possibilities in the library.

A number of commercial operating systems for embedded systems sprang to life in the 1980s. This primordial stew has evolved to the present-day stew of commercial operating systems. Today, there are a few dozen viable commercial operating systems from which to choose. A few big players have emerged, such as VxWorks, pSOS, Neculeus and Windows CE.

Many embedded systems do not have any operating system at all, just a control loop. This may be sufficient for very simple ones; however, as systems grow in complexity, an operating system becomes essential or the software grows unreasonably complex. Sadly, there are some horribly complex embedded systems that are complex only because the designers insisted they did not need an operating system.

Increasingly, more embedded systems need to be connected to some sort of network, and hence, require a networking stack. Even the doorknob in many hotels has an embedded microprocessor connected to a network.

For simple embedded systems that are just coded in a loop, adding a network stack may raise the complexity level to the point that an operating system is desirable.

In addition to a variety of commercial operating systems, there is an amazing number of proprietary operating systems. Many of these are created from scratch, such as Cisco's IOS; others are derived from some other operating system. For example, many network products are derived from the same version of the Berkeley UNIX system, because it has complete networking capability. Others are based on public domain operating systems such as KA9Q from Phil Karn.

Linux as an embedded OS is a new candidate with some attractive advantages. It is portable to many CPUs and hardware platforms, stable, scalable over a wide range of capabilities and easy to use for development.

Tools—Breaking the ICE Barrier

A key element in developing embedded systems is the set of available tools. Like any craft or profession, good tools help to get the job done faster and better. At different stages of development, different tools may be required.

Traditionally, the first tool used to develop embedded systems was the in-circuit emulator (ICE). This is a relatively expensive piece of equipment that typically hacks into the circuitry between the microprocessor and its bus, which allows the user to monitor and control all activity in and out of the microprocessor. These can be difficult to set up, and because of their invasive nature, can provoke erratic performance. However, they give a very clear picture of what is happening at the bus level, and eliminate a lot of guesswork at the very lowest level of the hardware/software interface.

In the past, some projects relied on this as the primary debugging tool, often through all stages of development. However, once the initial software works well enough to support a serial port, most debugging can be done without an ICE using other methods. Also, most newer embedded systems use a fairly cookbook microprocessor design. Often, corresponding working startup code is available that can be used to get the serial port working in short order. This means that one can often get along quite nicely without an ICE. Eliminating the ICE stage lowers the cost of development. Once the serial port is up, it can be used to support several layers of increasingly sophisticated development tools.

Linux is based on the GNU C compiler, which, as part of the GNU tool chain, works with the **gdb** source-level debugger. This provides all the software tools

you need to develop an embedded Linux system. Here is a typical sequence of debug tools used to bring up a new embedded Linux system on new hardware.

1. Write or port startup code. (We will talk more about this later.)
2. Write code to print a string on the serial port, i.e., “Hello World”. (Actually, I prefer “Watson, come here I need you”, the first words spoken over a telephone.)
3. Port the gdb target code to work over the serial port. This talks to another Linux “host” system which is running the gdb program. You simply tell gdb to debug the program via the serial port. It talks over the serial port to the gdb target code on your test computer, giving you full C source-level debugging. You may also want to use this same capability to download the additional code into RAM or flash memory.
4. Use gdb to get the rest of the hardware and software initialization code to work, to the point where the Linux kernel starts up.
5. Once the Linux kernel starts, the serial port becomes the Linux console port and can be used for subsequent development. Use **kgdb**, the kernel debug version of gdb. Often, this step is not required. If you have a network connection, such as 10BaseT, you will probably want to get it working next.
6. Once you have a fully functional Linux kernel running on your target hardware, you can debug your application processes. Use either gdb or a graphical overlay on gdb such as **xgdb**.

Real Time—Says Who?

Simply put, the majority of real-time systems aren't. Embedded systems are often misclassified as real-time systems. However, most systems simply do not require real-time capabilities. Real time is a relative term. Purists will often define hard real time as the need to respond to an event in a deterministic manner and in a short time, i.e., microseconds. Increasingly, hard real-time functions in this tight time range are being implemented in dedicated DSP (digital signal processor) chips or ASICs (application-specific ICs). Also, these requirements are often simply designed out through the use of a deeper hardware FIFO, scatter/gather DMA engines and custom hardware.

Many designers agonize over the need for real-time performance without a clear understanding of what their real requirements are. For most systems, near real-time response in the one- to five-millisecond range is sufficient. Also, a softer requirement may be quite acceptable, something like:

> The Windows 98 Crashed_Yet monitor interrupt must be processed within 4 milliseconds 98% of the time, and within 20 milliseconds 100% of the time.

These soft requirements are much easier to achieve. Meeting them involves a discussion of context switch time, interrupt latency, task prioritization and scheduling. Context switch time was once a hot topic among OS folks. However, most CPUs handle this acceptably well, and CPU speeds have gotten fast enough that this has ceased to be a major concern.

Tight real-time requirements should usually be handled by an interrupt routine or other kernel context driver functions in order to assure consistent behavior. Latency time, the time required to service the interrupt once it has occurred, is largely determined by interrupt priority and other software that may temporarily mask the interrupt.

Interrupts must be engineered and managed to assure that the timing requirements can be met, just as with any other operating system. On Intel x86 processors, this job can be handled quite nicely by the real-time extension to Linux (RTLinux, see http://www.rtlinux.org/). This essentially provides an interrupt processing scheduler that runs Linux as its background task. Critical interrupts can be serviced without the rest of Linux knowing about them. Thus, you get a lot of control over critical timing. Interfaces are then provided between the real-time level and the basic-Linux level with relaxed timing constraints. This provides a real-time framework similar to other embedded operating systems. In essence, the real-time critical code is isolated and "engineered" to meet the requirement, and the results of this code are handled in a more generic manner, perhaps at the application task (process) level.

Embedded System—a Definition

One view is that if an application does not have a user interface, it must be embedded, since the user does not directly interact with it. This is, of course, overly simplistic. An elevator-control computer is considered embedded, but has a user interface: buttons to select the floor and an indicator to show on which floor the elevator is now located. For embedded systems connected to a network, this distinction blurs even further if the system contains a web server for monitoring and control. A better definition might focus on the intended functions or primary purpose of the system.

Since Linux provides both a basic kernel for performing the embedded functions and also has all the user interface bells and whistles you could ever want, it is very versatile. It can handle both embedded tasks and user interfaces. Look at Linux as a continuum: scaling from a stripped-down micro-kernel with memory management, task switching and timer services and nothing else, to a full-blown server, supporting a full range of file system and network services.

A minimal embedded Linux system needs just these essential elements:

- a boot utility
- the Linux micro-kernel, composed of memory management, process management and timing services
- an initialization process

To get it to do something useful and still remain minimal, you need to add:

- drivers for hardware
- one or more application processes to provide the needed functionality

As you add more capabilities, you might also need these:

- a file system (perhaps in ROM or RAM)
- TCP/IP network stack
- a disk for storing semi-transient data and swap capability

Hardware Platforms

Choosing the best hardware is a complex job and fraught with tar pits of company politics, prejudices, legacies of other projects and a lack of complete or accurate information.

Cost is often a key issue. When looking at the costs, make sure you look at total product costs, not just the CPU. Sometimes a fast, cheap CPU can become an expensive dog of a product, once you add the bus logic and delays to make it work with your peripherals. If you are a software geek, chances are the hardware decisions have already been made. However, if you are the system designer, it is your due diligence to make a real-time budget and satisfy yourself that the hardware can handle the job.

Start with a realistic view of how fast the CPU needs to run to get the job done —then triple it. It is amazing how fast theoretical CPU capacity disappears in the real world. Don't forget to factor in how your application will utilize any cache.

Also, figure out how fast the bus needs to run. If there are secondary buses such as a PCI bus, include them also. A slow bus or one that is saturated with DMA traffic can slow a fast CPU to a crawl.

CPUs with integrated peripherals are nice because there is less hardware to be debugged, and working drivers are frequently already available to support the popular CPUs. However, in my projects, these chips always seem to have the wrong combination of peripherals or don't have the capabilities we need. Also,

just because the peripherals are integrated, don't assume this leads to the cheapest solution.

Squeezing 10 Pounds of Linux into a 5-Pound Bag

One of the common perceptions about Linux is that it is too bloated to use for an embedded system. This need not be true. The typical Linux distribution set up for a PC has more features than you need and usually more than the PC user needs also.

For starters, let's separate the kernel from the tasks. The standard Linux kernel is always resident in memory. Each application program that is run is loaded from disk to memory where it executes. When the program finishes, the memory it occupies is discarded, that is, the program is unloaded.

In an embedded system, there may be no disk. There are two ways to handle removing the dependence on a disk, depending on the complexity of the system and the hardware design.

In a simple system, the kernel and all applications processes are resident in memory, when the system starts up. This is how most traditional embedded systems work and can also be supported by Linux.

With Linux, a second possibility opens up. Since Linux already has the ability to “load” and “unload” programs, an embedded system can exploit this to save RAM. Consider a typical system that includes a flash memory, perhaps 8 to 16MB of flash, and 8MB of RAM. The flash memory can be organized as a file system. A flash driver is used to interface the flash to the file system. Alternatively, a flash disk can be used. This is a flash part that emulates a disk to the software. One example of this is the DiskOnChip from M-Systems (http://www.m-systems.com/) which can support up to 160MB. All of the programs are stored as files on the flash file system and are loaded into RAM as needed. This dynamic “load on demand” capability is a powerful feature that makes it easier to support a range of features:

- It allows the initialization code to be discarded after the system boots. Linux typically uses a number of utility programs that run outside the kernel. These usually run once at initialization time, then never again. Furthermore, these utility programs can run sequentially, one after the other, in a mutually exclusive fashion. Thus, the same memory can be used over and over to “page in” each program, as the system boots. This can be a real memory saver, particularly for things like network stacks that are configured once and never changed.

- If the Linux loadable module feature is included in the kernel, drivers can be loaded as well as the application programs. The software can check the hardware environment and adaptively load only the appropriate software for that hardware. This eliminates the complexity of having one program to handle many variations of the hardware at the expense of more flash memory.
- Software upgrades are more modular. You can upgrade the application and loadable drivers on the flash, often while the system is running.
- Configuration information and runtime parameters can be stored as data files on the flash.

Un-Virtual Memory

Another feature of standard Linux is its virtual memory capability. This is that magical feature that enables application programmers to write code with reckless abandon, without regard to how big the program is. The program simply overflows onto the swap area of the disk. In an embedded system without a disk, this capability is usually unavailable.

This powerful feature is not needed in an embedded system. In fact, you probably do not want it in real-time critical systems, because it introduces uncontrolled timing factors. The software must be more tightly engineered to fit into the available physical memory, just like other embedded systems.

Note that depending on the CPU, it is usually advisable to keep the virtual memory code in Linux, because cutting it out entails quite a bit of work. Also, it is highly desirable for another reason—it supports shared text, which allows multiple processes to share one copy of the software. Without this, each program would need to have its own copy of library routines like **printf**.

The virtual-memory paging capability can be turned off simply by setting the swap space size down to zero. Then if you write programs that are bigger than actual memory, the system will behave the same way as it does when you run out of swap space; the program will not load, or perhaps a **malloc** will fail, if the program asks for too much memory.

On many CPUs, virtual memory also provides memory management isolation between processes to keep them from overwriting each other's address space. This is not usually available on embedded systems which just support a simple, flat address space. Linux offers this as a bonus feature to aid in development. It reduces the probability of a wild write crashing the system. Many embedded systems intentionally use “global” data, shared between processes for efficiency reasons. This is also supported in Linux via the shared memory feature, which exposes only the parts of memory intended to be shared.

File Systems

Many embedded systems do not have a disk or a file system. Linux does not need either one to run. As mentioned before, the application tasks can be compiled along with the kernel and loaded as one image at boot time. This is sufficient for simple systems. However, it lacks the flexibility described previously.

In fact, if you look at many commercial embedded systems, you'll see that they offer file systems as options. Most are either a proprietary file system or an MS-DOS-compatible file system. Linux offers an MS-DOS-compatible file system, as well as a number of other choices. The other choices are usually recommended, because they are more robust and fault-tolerant. Linux also has check and repair utilities, generally missing in offerings from commercial vendors. This is especially important for flash systems which are updated over a network. If the system loses power in the middle of an upgrade, it can become unusable. A repair utility can usually fix such problems.

The file systems can be located on a traditional disk drive, on flash memory, or any other media for that matter. Also, a small RAM disk is usually desirable for holding transient files.

Flash memories are segmented into blocks. These may include a boot block containing the first software that runs when the CPU powers up. This could include the Linux boot code. The rest of the flash can be used as a file system. The Linux kernel can be copied from flash to RAM by the boot code, or alternatively, the kernel can be stored in a separate section of the flash and executed directly from there.

Another interesting alternative for some systems is to include a cheap CD-ROM drive. This can be cheaper than flash memory, and supports easy upgrades by swapping CD-ROMs. With this, Linux simply boots off the CD-ROM and gets all of its programs from the CD-ROM in the same way it would from a hard disk.

Finally, for networked embedded systems, Linux supports NFS (Network File System). This opens the door for implementing many of the value-added features in networked systems. First, it permits loading the application programs over a network. This is the ultimate in controlling software revisions, since the software for each embedded system can be loaded from a common server. It is also useful, while running, to import and export a plethora of data, configuration and status information. This can be a very powerful feature for user monitoring and control. For example, the embedded system can set up a small RAM disk, containing files which it keeps updated with current status information. Other systems can simply mount this RAM disk as a remote disk over the network and access status files on the fly. This allows a web server on

another machine to access the status information via simple CGI scripts. Other application packages running on other computers can easily access the data. For more complex monitoring, an application package such as MatLab (http://www.mathworks.com/products/matlab/) can easily be used to provide graphical displays of system operation at an operator's PC or workstation.

Booting—Where's LILO and the BIOS?

When a microprocessor first powers up, it begins executing instructions at a predetermined address. Usually there is some sort of read-only memory at that location, which contains the initial start-up or boot code. In a PC, this is the BIOS. It performs some low-level CPU initialization and configures other hardware. The BIOS goes on to figure out which disk contains the operating system, copies the OS to RAM and jumps to it. Actually, it is significantly more complex than that, but this is sufficient for our purposes. Linux systems running on a PC depend on the PC's BIOS to provide these configuration and OS-loading functions.

In an embedded system, there often is no such BIOS. Thus, you need to provide the equivalent startup code. Fortunately, an embedded system does not need the flexibility of a PC BIOS boot program, since it usually needs to deal with only one hardware configuration. The code is simpler and tends to be fairly boring. It is just a list of instructions that jam fixed numbers into hardware registers. However, this is critical code, because these values need to be correct for your hardware and often must be done in a specific order. There is also, in most cases, a minimal power-on self-test module that sanity-checks the memory, blinks some LEDs, and may exercise some other hardware necessary to get the main Linux OS up and running. This startup code is highly hardware-specific and *not* portable.

Fortunately, most systems use a fairly cookbook hardware design for the core microprocessor and memory. Typically, the chip manufacturer has a demo board that can be used as a reference design—more or less copying it for the new design. Often, startup code is available for these cookbook designs, which can be modified for your needs fairly easily. Rarely will new startup code need to be written from scratch.

To test the code, you can use an in-circuit emulator containing its own “emulation memory”, which replaces the target memory. You load the code into the emulator and debug via the emulator. If this is not available, you may skip this step, but count on a longer debug cycle.

This code ultimately needs to run from some non-volatile memory, usually either flash or EPROM chip. You will need some way to get the code into this chip. How this is done will depend on the “target” hardware and tools.

One popular method is to take the flash or EPROM chip and plug it into an “EPROM” or “flash burner”. This will “burn” (store) your program into the chip. Then, plug the chip into a socket on your target board and turn on the power. This method requires the part to be “socketed” on the board; however, some device package formats cannot be socketed.

Another method is via a JTAG interface. Some chips include a JTAG interface which can be used to program the chip. This is the most convenient way to do it. The chip can be permanently soldered onto the board, and a small cable run from a JTAG connector on the board, usually a PC card, to a JTAG interface. The downside is some custom programming is usually required on the PC to operate the JTAG interface. This same facility can also be used in production for smaller-quantity runs.

Robustness—More Reliable than a Politician's Promise

Linux is generally considered to be very reliable and stable when running on PC hardware, particularly when compared to a popular alternative. How stable is the embedded kernel itself? For most microprocessors, Linux is quite good. A Linux kernel port to a new microprocessor family is usually done to more than just the microprocessor. Typically, it is ported to one or more specific target boards to which Linux is ported. These boards include some specific peripherals as well as the CPU.

Fortunately, much of the kernel code is processor-independent, so porting concentrates on the differences. Most of these are in the memory management and interrupt handling areas. Once these are ported, they tend to be fairly stable. As discussed before, boot strategies vary depending on the hardware specifics, and you should plan on doing some customization.

The device drivers are more of a wild card: some are more stable than others. Also, the selection is rather limited; once you leave the ubiquitous PC platform, you may need to create your own. Luckily, many device drivers are floating around, and you can probably find one close to what you need that can be modified. The driver interfaces are well-defined. Most drivers of a like kind are fairly similar, so migrating a disk, network or serial port driver from one device to another is usually not too difficult. I have found most drivers to be well-written and easy to understand, but keep a book on the kernel structures handy.

In my experience, Linux is at least as stable as the big-name commercial operating systems with which I have worked. Generally, the problems with these operating systems and Linux stem from a misunderstanding of the subtlety of how things work, rather than hard coding bugs or basic design errors. Plenty of war stories abound for any operating system and need not be

repeated here. The advantage to Linux is that the source code is available, well-commented and very well-documented. As a result, you are in control of dealing with any problems that come up.

Along with the basic kernel and device drivers, some additional issues arise. If the system has a hard disk, the reliability of the file system comes into question. We have over two years of field experience with an embedded Linux system design employing a disk. These systems are almost never shut down properly. Power just gets disconnected at random times. The experience has been very good, using the standard (EXT2) file system. The standard Linux initialization scripts run the **fsck** program, which does an excellent job of checking and cleaning up any dangling inodes. One change that may be wise is to run the **update** program at a 5 or 10-second interval instead of the default 30 seconds. This shortens the time window that data sits in the local memory cache before being flushed to disk, thus lowering the probability of losing data.

Where Are the Skeletons?

Embedded Linux does have its drawbacks. For one, it can be a memory hog, although it's not much worse than some of its commercial competitors. Part of this can be whittled down by removing unused features, but this may take more time than it is worth and may induce some ugly bugs if not done carefully.

Most Linux applications are addicted to virtual memory, which is of dubious value in many embedded systems, so don't assume an embedded system with no disk will run any Linux application.

The kernel-level debug tools are not all that great, particularly at the lower levels. **kgdb** can have hard landings fairly easily, and you just have to reboot. Unfortunately, print statements get used more than we'd like.

The worst problem for me, however, is a psychological one. Linux is extremely flexible. Embedded systems are generally not flexible; rather, they are highly engineered to perform their intended function optimally. There is a tendency to preserve the flexibility, keep the general purpose capabilities and make as few changes as possible. These are noble goals, but they come at the expense of sharply tuning the system for the specific job. Keeping this flexibility can result in extra work and carrying extra software baggage around, and sometimes involves tradeoffs which degrade performance. One example that comes up repeatedly is configuration. Consider configuring the IP address on a network interface, which is usually done by running the **ifconfig** program from a startup script. This is a 28K program that could be replaced with a few lines of code to initialize the appropriate structures, using data from a configuration file.

Nevertheless, while this is a reasonable thing to do, it still “hurts” because it is contorting the software in ways it was never meant to be used.

The Bottom Line

Using Linux for an embedded system is possible and has been done. It works. It is reliable. The development costs are in line with the alternatives.

Glossary

Joel R. Williams is the principal architect at emLinux. He develops embedded systems for networking applications and telecommunications. He can be reached at joel@emlinux.com, or visit http://www.emlinux.com/.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Porting Linux to a PowerPC Board

He Zhu

Xiaoqiang Chen

Issue #66, October 1999

An experiment and experience in using Linux in an embedded application.

We believe Linux is going to play a significant role in embedded applications. It is compliant with POSIX 1003.1 and supports the POSIX soft real-time extension. Theoretically, it is capable of supporting a wide range of embedded applications which require only soft real-time performance such as Internet routing. Easy customization makes it even more attractive. To investigate the possibilities of using Linux as a platform for embedded systems, we conducted an experiment in which we ported the kernel to a PowerPC-based board. The porting took a few weeks. Our result is a Linux port, called **elinux** here, based on the Linux kernel 2.1.132 which was the latest version at the time we began the experiment. **elinux** runs dozens of commands and programs, including **bash** and **vi** on the console via a serial port. Porting Linux is actually a very enjoyable experience. Besides, all the work is based on open-source software exclusively.

Introduction

Linux is portable. We have already seen many Linux ports on various processors. However, few documents describe how to port Linux. The information is scattered in various documents and source code. Porting is not an easy job for any operating system. Even a minor change in the kernel code to suit a particular piece of hardware needs a considerable effort. Fortunately for Linux, all major components of the kernel are already designed to be architecture-independent. This makes the work relatively much easier.

It is harder to port Linux to a board with a new processor than with a processor Linux has already supported. In the latter case, we can reuse board-independent code; for example, the code for memory management. Only a

relatively small portion of the kernel code is board-dependent. When we considered implementing elinux, we tried to avoid reinventing the wheel. We kept most of the necessary changes limited to board-dependent parts. Our experiment was done on a PowerPC-based board. Linux already has ports for PowerPC-based machines such as Power Macintosh and a few PowerPC-based embedded boards. However, due to diverse board architectures, configurations and booting methods, modifications are required when we consider a new board.

In our case, some changes in the kernel and a few small new programs were created to support elinux. In the following article, we emphasize our experience with the issues of most concern in our work rather than implementation details. These include setting up the cross-development platform, designing the booting sequence, modifying the kernel, creating the executable image and root file system image and debugging.

Experimental Hardware

Our goal is not to port Linux to a particular kind of hardware. Instead, we are interested in approaches to porting the Linux kernel to potential embedded systems. Thus, it doesn't matter what kind of board we choose as long as it presents a typical situation. The board we actually used in our experiments is a PowerPC-based board built on a PCI bus. It has a PowerPC 603e processor, an MPC 106 as the memory controller and the PCI bridge, 32MB DRAM memory, a PC16552 DUART chip for two serial ports, a memory-mapped real-time clock in non-volatile memory and a simple but customized interrupt controller. It also has two flash memory slots and an Intel 82558 LAN controller to provide three LAN ports.

The board has its own bootstrap code in ROM. This code does hardware initialization and also provides a simple native file system and the TFTP support. Although we used these two services to boot elinux, our approach can support booting completely from ROM.

Cross-Development Platform

Simply installing binaries from any Linux distribution does not guarantee a working cross-development platform. Some people experienced difficulties in setting up a complete cross-development environment. Our experience shows that this is not only possible, but also brings us a fair amount of convenience, as we can use some of the most popular software packages. Our lessons are proper distributions, proper configurations and recompiling.

Here are our actual steps in setting up the cross-development platform for PowerPC on Pentium machines:

- Install Red Hat 5.2 on a Dell OptiPlex Pentium II 400 MHz PC with 256MB RAM and 8GB SCSI hard disk.
- Get the source code of the latest stable Linux kernel, 2.0.36 when our work began.
- Recompile the kernel from the kernel source to make sure support for loopback devices, RAM disks and other necessary items is included.
- Use the newly recompiled image to boot the development platform.

After the base development system is ready, we install cross-development support as follows:

- First, install the source code of the binary utilities binutils-2.9.1.0.15 which includes cross-assembler, cross-loader and other cross-utilities.
- Recompile and install the cross-utilities for PowerPC with Linux.
- Install the source code of the compiler gcc 2.8.1.
- Recompile and install for the cross-compiler.

After this, the environment is ready to develop elinux on the Pentium machine. As for the standard C library glibc, we don't need it at this time. If we wanted to do general programming for elinux, we could cross-compile the glibc. For convenience in testing small programs, we actually used glibc binaries for PowerPC from the Red Hat distribution.

Design of the Booting Sequence

Booting an operating system seems easy. You just power on the system, and after a while, you'll see a prompt on the console which indicates the system is running. If we look into the booting internals, we get a more complicated view. Booting includes hardware initialization and software startup, operations which differ from board to board. Different booting code exists for each kind of board in the Linux architecture-dependent source code directory, that is, linux/arch/. For a new board, we usually have to add a new booting sequence.

A typical embedded board has no floppy and no hard disk. Code and data are initially put in ROM or can be downloaded through a network connection. We have designed a general approach to booting such a system from ROM.

Our approach is to divide the booting into two stages supported by two separate loaders. One is called the image loader **iloader**, and the other is the Linux kernel loader **kloader**. **iloader** is ROM-able. That is, after the system is powered up, it starts, does necessary hardware initialization, then moves the

Linux kernel loader from ROM to the proper location in RAM. **kloader** starts running once iloader finishes. First, it does more hardware initialization. Then, it sets up the environment for booting the Linux kernel by uncompressing the Linux kernel image. Finally, it jumps to the kernel code to begin the main Linux startup sequence.

To make things clearer, consider elinux. The final elinux image, which we call an elinux ball, is packed as a single file containing three items:

1. statically linked iloader executable binary
2. our zImage consisting of the uncompressed kernel image vmlinux.bin.gz plus kloader
3. compressed root file system image ramdisk.gz

The size of an elinux ball depends on how many services and programs are included. In our experiments, it is limited to 2MB, which is big enough for most situations. If larger programs are needed, they can be downloaded after the system is up. It is good to keep the ball small. The packing is simply done by a tool called **packbd** (packing binaries and data images). The elinux ball is obtained using the command:

```
packbd iloader kloader vmlinux.bin.gz ramdisk.gz\
elinux
```

iloader is the entry point to start the elinux ball. Because it is ROM-able, the whole elinux ball is also ROM-able. However, we don't have to put it into ROM to boot the system. Actually, in our development, we use the native networking service TFTP to download the elinux ball into a RAM area and start execution.

The implementations of iloader, kloader and packbd are straightforward for any system, except for hardware initialization which usually requires more effort.

Modifying the Kernel

Kernel modification is the hardest part of porting. Fortunately, Linux has been designed to be portable, with its sources well-organized into a tree structure. Once you have made considerable investigations into the kernel sources, the things to do become clear. As we mentioned earlier, changes and even new pieces of code are required when porting Linux to a new board. Basically, all board-dependent code must be modified or adjusted even if we use a Linux-supported processor. Changes are also needed when we have new requirements or any bug fixes. Most are concentrated in a few files, so hopefully this can help us conveniently catch up with new releases.

We used the experimental kernel version 2.1.132 PPC port for elinux. Almost all changes are limited to the board-dependent parts, that is, in the subdirectory linux/arch/ppc for our PowerPC board. Dozens of changes have been made; many for adapting to the new hardware, other things for bug fixes and new requirements such as new memory mapping.

Major changes for elinux include those for hardware initialization, PCI bus initialization, memory management, timer processing and interrupt processing.

Hardware initialization is the ugliest part in elinux. The loader iloader should do the most essential part of the job, such as initialization of the memory controller and PCI controller. However, our implementation of iloader simply ignores this part because we find it is unnecessary to do it again after the board's ROM code has done it. Of course, iloader has to do it if iloader is the first code to run from ROM. The kernel initialization does others such as memory protection and bus device initialization.

The original PPC port talks with PCI devices through interfaces with the BIOS. For our board, we assume there is no similar thing. We just leave the interfaces empty at the beginning. Whenever we add a new PCI device, we write code directly to set up the related base addresses, IRQs and access methods.

A few considerations are necessary for memory management. Although we don't need any changes in the major parts for memory management such as virtual memory management and paging, we do have modification requests. They are mainly for setting up the particular memory sizes and ranges, re-arranging the memory during the kernel startup, the use of PowerPC BAT (Block Address Translation) register pairs and memory mappings between physical and virtual addresses.

For timer processing, two things are modified. One is to adjust the parameters to set the PowerPC's decrementer to suit the board's bus rate. This decrementer is used to generate a timer interrupt every jiffy time (10 milliseconds). The other change is to provide the interface with direct access to the Real Time Clock (RTC) on board.

Another major change is for the interrupt controller. This controller is simple and controls only 16 IRQs through a status register, a mask register and a latch register. All the registers are 16 bits. Each bit corresponds to one IRQ. New simple code is added to handle it.

We have successfully relocated the kernel in the virtual space by redefining the symbol **KERNELBASE** as a Makefile macro. This involves a few changes in the kernel initialization code. Since we are able to relocate the kernel, we can

reserve the space for some special purposes. For example, to load the kernel at the address 0xa0000000 instead of the default address 0xc0000000, we just define **KERNELBASE** in the top Makefile this way:

```
KERNELBASE = 0xa0000000
```

Minor changes are made in various Makefiles. These are necessary because we need new rules to create the elinux ball, we have a few new files to compile and link, and we found bugs in the Makefile when doing cross-compiling.

As for device drivers, we are concerned only with the serial driver for the console port. Other drivers may be added later if necessary, such as a LAN driver and drivers to control customized devices. During our experiments, we communicate with elinux using **minicom** over a serial port. We use the serial driver from the Linux code drivers/char/serial.c. A minor change is made for adjusting the baud rate, and another change is made in its header file since the serial port has a different IRQ number.

After all is done properly, we see through the console that elinux starts and runs happily.

```
ELINUX VERSION 0.001 March 1999
 Start booting Linux on Experiment Board ...
 ... (omitted long booting messages)
 #      (we start ash after the kernel is up)
```

Creating the Kernel Image and the Root Image

As discussed previously, the elinux ball needs a statically linked kernel image and a root file image. Their preparation needs some skills.

The Linux source provides the rules to create various kernel images for different platforms. For our system, we need a compressed binary kernel image, vmlinux.bin.gz. The steps to create it are, in order, configuration, compiling and linking, transferring to binary format and compressing. In configuration, make sure the RAM disk support and the initial root file system support are selected, and that all unnecessary options are disabled. Compile and link the kernel for the ELF executable called **vmlinux**. Then transfer to binary format and compress it by commands such as:

```
(CROSS_COMPILE)objcopy -S -O binary vmlinux\
vmlinux.bin
gzip -vf9 vmlinux.bin
```

Because we choose to mount an initial root file system in RAM after the kernel is up, we have to prepare a root image, called ramdisk.gz, and put it into the elinux ball. We do this by creating an EXT2 file system in a 4MB RAM disk on the cross-development platform. Next, create the subdirectories such as /etc, /dev,

/bin and /lib. Then, copy scripts, binaries, device nodes, etc. onto the RAM disk. Finally, compress the RAM disk image and get ramdisk.gz. For example, to create a RAM disk in /dev/ram1, type:

```
rdev -r /dev/ram1 4096
```

Make a file system and mount it to /tmp by typing:

```
mke2fs -vm0 /dev/ram1 4096
mount /dev/ram1 tmp
```

To create a device, use **cp -d** or **mknod** in this way:

```
mknod ttyS0 c 4 64
```

This creates a device node for the serial console port on elinux, with major number 4 and minor number 64.

After everything in /tmp is ready, compress it by typing:

```
dd if=/dev/ram1 bs=1k count=4096 |
gzip -v9 > ramdisk.gz
```

What should be included in the initial root RAM disk will depend on our requirements. We copy a minimum number of shared libraries, plus some programs like bash and vi for tests.

Debugging Support

Bugs are inevitable. “Given enough eyeballs, all bugs are shallow” is only partly true. Most often, we have to debug without outside help. Debugging may be painful, especially for system booting. There is a booting debugging tool supported by special hardware, but we don't want to rely on it. Other debugging mechanisms such as **printk** and **gdb** help in many cases, but they need too many system services. For example, the Linux kernel debugging support, printk, works only after the system is ready to write to a console or the file system. If the system crashes before that time, we get nothing from printk, even if printk uses a memory buffer to store information earlier in the process.

Simplicity means efficiency in this case. To help solve the problem, we add **rprintf**. It is a simple printing function using raw output, which writes characters directly to the console I/O port without any buffering and any other support. **rprintf** is like printf, but based only on this kind of raw output. It works very soon after the iloader runs, so it can be used to debug kloader and the Linux kernel as well. **rprintf** helps us solve most problems in the early stages of booting. We did have a few problems before rprintf is initialized, but we are not helpless. Our suggestion is to insert an operation to force a system reboot; in this way, you can locate the problem soon. We assume you know when the

board starts rebooting. We provide a function called **rreboot** to do this job for our board by simply jumping to the system reboot entry point in ROM.

Concluding Remarks

Unlike other projects, this work relies heavily on the Internet. We have learned much about Linux and obtained resources such as the kernel code and Linux-related documentation from the Web. To do something meaningful on Linux, follow proven patterns by utilizing open-source code as much as possible. Reuse pieces of code that run, even if changes are required. Don't be too ambitious in the beginning. Spend plenty of time on investigation before moving on to the next stage. Also, take the time to have a good design at the beginning and choose good debugging support. Always refer to Linux books and web sites first whenever you need help (see Resources). Readers can easily discover a huge amount of related information on the Web.

The same thing can be done in several ways. Our experiments are far from comprehensive, but we are confident in Linux's potential in some embedded systems. We hope our experience will help other people who want to port a Linux kernel to their embedded system. There are many related issues for us to research—much fun is ahead.

Resources

He Zhu received his Ph.D. in computer science at the University of Manchester, England and is currently a researcher at Bell Labs. His interests include networking and system software. He can be reached at zhuhe@dnrc.bell-labs.com.

Xiaoqiang Chen received his Ph.D. in computer science from Cambridge University, England. He is currently a technical manager with Bell Laboratories, Lucent Technologies, where he has conducted research and development in high-speed networks. He can be reached at xchen@bell-labs.com.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Embedding Linux to Control Accelerators and Experiments

Andy Gotz

Petri Makijarvi

Bernard Regad

Manuel Perez

Paolo Mangiagalli

Issue #66, October 1999

A scientific laboratory in Europe depends on Linux for controlling equipment used in their research.

Linux is being used at the European Synchrotron Radiation Facility to build distributed embedded controllers. The embedded systems are either PC/104-based systems, which boot from a flashdisk, or VME crates which boot from the network. The devices being controlled vary from serial lines to stepper motors to CCD cameras. The control software is written using an object-oriented toolkit that we developed, called TACO (Telescope and Accelerator Control Objects). Using TACO all control points are implemented as device objects. This article will describe how we have implemented embedded controllers using Linux and present some examples.

Introduction

What is an embedded controller? As the name indicates, an embedded controller is a mixture of control hardware and software which is embedded, i.e., integrated, into the equipment it is supposed to control. Examples of where embedded controllers can be found are abundant in daily life: printers, portable telephones, the brakes of your car, etc. The requirements for embedded controllers are usually that they be physically small, have a small memory footprint, low power consumption and low cost. Most of these requirements are dictated by the fact that controllers will be built into hardware systems and in large production volumes.

Figure 1. Aerial view of the ESRF nestled between the Drac and Isere rivers at the foot of the French Alps (Grenoble).

Figure 1

At the European Synchrotron Radiation Facility (ESRF, see Figure 1), our main job is to connect a wide variety of control hardware with an equally large number of different interfaces. We are faced with the task of interfacing hundreds of power supplies to control the accelerator magnets which guide the electrons, thousands of stepper motors which move and position the experiments on the beamlines and a myriad of other devices. Most of these devices contain embedded controllers which export their functionality via a dedicated interface such as a serial line, parallel interface or computer bus to the outside world. Our task consists of determining how to interface these different hardware devices in a coherent and efficient manner so that higher-level software (often not written by us) can access it easily.

The European Synchrotron Radiation Facility

The ESRF, located in Grenoble, France, is a multinational research institute supported by 15 participating countries. The ESRF is operated as a non-profit enterprise under French law. Management is supervised by a Council whose delegates are designated by the member parties.

This large experimental facility performs basic and applied research in physics, chemistry, material and life sciences. The research is facilitated through the use of a powerful source of radiation in the X-ray range. This synchrotron radiation, used at experimental stations called beamlines, has remarkable capabilities. Pushing the technological limits, the ESRF performs novel experiments which have not been feasible before.

The construction of the ESRF started in 1988. The inauguration and opening of the first 15 beamlines to scientific users took place in September 1994. Currently, 40 beamlines are operating 24 hours a day, 7 days a week. More information about the ESRF can be found on their web site at http://www.esrf.fr/.

Our answer to the interface problem has been to use Ethernet and the TCP/IP protocol as the ubiquitous interface for all devices. We have developed an object-oriented toolkit called TACO for wrapping all devices and then exporting them to the external world (our users) as objects on the network, accessible via an application programmer's interface (API). In order to wrap the different pieces of hardware, we often build embedded controllers close to the hardware which run the TACO wrapper software and make the hardware available via the network.

We will describe two types of embedded controllers which we build using Linux —PC/104-based controllers and VME-based controllers.

Why Linux?

Why use Linux for embedded controllers? Although the choice to use Linux is obvious to many readers, the reasons are not always the same. In our case, we needed an operating system with the following features: highly configurable, excellent TCP/IP stack implementation, easy to program when it comes to writing device drivers, modern programming standard (e.g., POSIX, CORBA, Java, HTTP) support, stable, well-supported and not too expensive. Linux fulfills all these requirements and more. The fact that Linux is free and comes with the source makes it even more attractive.

Before Linux, we used commercial operating systems for embedded controllers and after two years of working with Linux, we noticed the difference. Gone are the days of flaky TCP/IP implementations, silence as an answer to our bug reports (now that we have the source, we can even fix the bugs ourselves), expensive contracts, lack of modern products and no one to talk to about our work. The choice of Linux for our embedded systems has meant that we now have the same OS from the low level to the desktop to our Beowulf cluster.

Choosing Linux does not mean that all questions have been answered, however. Questions remain, such as “When will industry embrace Linux?” and “How long will Linux last?” The answer to the first one is soon—we hope. The answer to the second is not forever, of course. As with any product/phenomenon, Linux will not last forever. At the ESRF, we renew our control systems roughly every ten years. It is important that Linux continues to exist for this time. In the worst-case scenario, we still have the source.

Embedding Linux

Having chosen Linux as our OS, we chose VME (for historical reasons) and PC/104 for our hardware. Strictly speaking, VME is not an embedded controller. It is a bus-based system which offers the ability to plug many I/O cards into a single crate. The I/O cards can communicate with the hardware without being aware of any other cards in the crate, or they can rely on other cards to do part of the work. This is one of the strengths of VME.

Why talk about VME if it is not a true embedded controller? Because we configure our VME systems like embedded controllers. They are totally diskless, i.e., they have only RAM on board, boot a generic image from the network, and in fact, resemble a general purpose I/O embedded controller (see Figure 2).

PC/104 is, however, a true embedded controller. It is a small-format PC which can be stacked together with I/O cards to form a dedicated controller. It is low-cost, low in power consumption, small and ideal for performing dedicated tasks. How do we configure Linux to boot and run on these two formats? Read on....

Figure 2. Paolo standing next to a medium-sized TacoBox with a PC-104 CPU inside being used as a dedicated SCSI controller to control a data acquisition system. The beast in the background is a robot for manipulating silicon wafers in the x-ray beam.

Figure 2

PC/104 alias TacoBox

The TacoBox project was started at the ESRF in 1995 with the goal of creating a low-cost, networked I/O device controller for distributed control systems. TacoBox is a very simple device with an I/O interface for the physical device to be controlled. The I/O interface is specified by the TACO I/O class as a set of commands possible to use with the physical device. Imagine a TacoBox as a black box which speaks TACO protocol as input and process I/O as output. The user interface is accessed via Ethernet using two major protocols. The configuration and (some) maintenance operations of the I/O interface can be done using an ordinary web browser.

We use the PC/104 form factor CPU and I/O boards to build TacoBoxes. A PC/104 is essentially an IBM-compatible PC in an embedded, industrial format. It is built around a 104-pin stack-through connector which is electrically compatible with the 8- and 16-bit ISA bus (PC-XT and PC-AT). The PC/104+ has an additional 32-bit bus, electrically compatible with the PCI bus. The PC/104 form factor is small, 90 x 95 mm. It is stackable, which means you can build a PC/104 system in a very small space. In a recent survey, PC/104-based systems were the third best-selling embedded bus system standard after VME and CompactPCI. The first prototypes of PC/104 ran under commercial real-time operating systems (RTOS) with limited on-board resources. When we got interested in Linux, we immediately wanted to install it on TacoBox. At first, we started with the same configuration as we used with those RTOS installations. We learned quite a bit about the different embedded Linux distributions out there. But we also learned that with just a little more money, we could get a Pentium-90 level PC/104 Single Board Computer (SBC) with 20MB of DRAM, IDE-interface, plus Ethernet (we used JUMPTec's MOPS/586). Add a hard disk and floppy drive, a SVGA card, a screen, a keyboard and a mouse and you have an “old-fashioned” desktop PC. On a desktop PC, you can install Linux without any problem, thanks to some great Linux distributions. At the ESRF, SuSE Linux 5.3 was installed on our desktop machines. Within a few hours, we had a desktop PC/104.

We fit all that in a box by removing the spinning hard disk first and replacing it with a solid-state, IDE-compatible, 24MB FlashPROM hard disk. It piggybacks neatly on the SBC. You do not want to write on a FlashPROM too often, because it is slow and will not support many continuous writing operations. Therefore, the root file system on the FlashPROM is mounted read-only and all the live files and directories are placed on the initial RAM disk, the /initrd. How do you fit a SuSE Linux distribution in 24MB? You give up Perl, Apache and man pages to get to that size. Again, it is a question of money, since FlashPROM disks with up to 200MB are available. What about the SVGA, the screen and other desktop decoration? Configure a serial line system console, and rip them all out.

VME

A farm of 250 VME bus-based systems is running here. All are running diskless and booting from a UNIX machine using BOOTP/TFTP protocols. The VME Single Board Computers used at the ESRF (see Figure 3) are MVME-167, with 8, 16 or 32MB of memory, and MVME-162-522, with 8 or 16MB of memory. When we wanted to install Linux on this platform, we looked for information on the Internet and were pleasantly surprised to find Richard Hirst's site about Linux for 680x0-based VME boards. What a relief to find such professional support for what we thought would be a major job for months to come.

Figure 3. Example of diskless VME crates used as general purpose input/output boxes. The lower crate (with the box of tacos in it) is running Linux and being used to control a high-resolution encoder card. The upper crate is running a commercial OS to control a variety of input/output cards. Work is in progress to port all the software from the commercial OS to Linux while staying in VME format.

Figure 3

Richard's distribution is for disk-based systems, so he helped us sort out some problems with the NFS-root-based systems. A very useful package that we added on our systems is Nick Holgate's TFTPLILO package, which allows us to have just one Linux boot image for all our Linux/68k VME systems.

We keep boot images for Linux/68k VME systems, as they are used on our FTP server (ftp://ftp.esrf.fr/pub/cs/ess/linux/). This allows all our collaborators to test device drivers and other software in a similar configuration as ours. It also allows any owner of a MVME-167 or a MVME-162-522 board to try Linux/68k without any investment or other hassle.

If you have already jumped on your keyboard, you may have noticed we are using the Debian file system and the 2.0.33 kernel. The good news is that

Debian now includes support for MVME-16x, MVME-17x and BVM VME bus Single Computer Boards.

Writing a device driver for a VME bus-based Linux computer requires knowing something about the VME bus and its principles of operation. Briefly, VME bus is an asynchronous bus that allows multiple bus masters, but (luckily) only one arbiter. It has seven daisy-chained interrupt lines that on most SBCs can be re-mapped to different local interrupt levels. The access to VME bus I/O boards is entirely memory-mapped, again with some physical translation between the local memory address and the VME bus address. All this is usually managed by some complex chips, such as Vmechip2 from Motorola. Writing a device driver for VMEbus requires some understanding of the interface chip. Our ftp site contains examples of general purpose Vmechip2 device drivers. It is good idea to read through those drivers before moving ahead to more complex device drivers.

TACO

To provide a unified software interface to many different pieces of hardware with as many different kinds of hardware interfaces, we have developed an object-oriented toolkit called TACO for doing just that. It is also used at the Hartbeesthoek Radio Astronomy Observatory in South Africa (see Resources) for controlling a radio telescope, and will be used at the FRM II research reactor in Germany for controlling future experiments.

TACO unifies the software interface with the hardware and the way control software is written by providing a framework for developing control objects. Each control point is implemented in a Device class as a method. At runtime, the Device classes create as many copies of Device objects as there are pieces of hardware, and export their functionality onto the network. The Device objects are served by Device servers, which are stand-alone processes under Linux. Clients (which can be graphical GUIs or other Device objects) access the Device objects via the Device Server API (DSAPI).

The network addressing is managed by a database (implemented with **gdbm**) and the network protocol by the ONC RPC (from Sun, which is now part of glibc). RPCs make network calls look like local procedure calls. The ONC RPC is the basis of NFS. This means it can be found on all systems where NFS has been ported, i.e., just about everywhere. It requires one extra process to be running per host—the portmapper, which manages the mapping of program number to local port numbers. Performance overhead induced by an RPC over the network is on the order of a few milliseconds. The memory footprint is less than 100 kilobytes per server, linked with shared libraries (TACO API's, glibc and pthreads).

The DSAPI implements various flavours of network calls needed for doing controls—synchronous (client blocks waiting for the answer), asynchronous (client continues immediately and retrieves answer later) and events (client registers interest in an event and gets sent events asynchronously). It also implements a large number of standard types, timeouts, UDP and TCP protocols and stateless connections. Device classes can be written in C or C++. The database is used to store and retrieve device-dependent information. Interfaces to high-level scripting languages such as Tcl, LabView, SPEC and Mathlab are available. Clients can also be written in C, C++ or Java.

Because of its simple approach, TACO scales easily. It has been used for laboratory systems consisting of a single device right up to an entire synchrotron which consists of more than 10,000 devices. We use TACO to control everything from simple digital I/O to entire data acquisition systems.

We are presently working on the next generation of TACO called TANGO (TAco Next Generation Objects). In TANGO, CORBA will replace ONC RPC, and it will be possible to write Device classes in Java as well as C++.

Examples of Embedded Controllers

We started using Linux in embedded controllers approximately one year ago. To date, we have built the following controllers.

Serial Line Controller

We have traditionally used a VME-based board with 16/18 ports to control RS-232 and RS-422 serial lines. We have thousands of serial lines today to control vacuum devices, power supplies, PLCs, etc. As the number of devices requiring serial lines increases each day (and in some crates, we have reached the maximum number of serial line VME boards), we started looking for a new system, which would be cheaper than the existing VME-based system and would be PC-based. (Many of the serial line devices are certified only on PCs.) That is how we came to the PC/104. Cost is not the only reason for changing: we also wanted to implement new technologies, like web support, for better maintenance and configuration of the serial lines. We wanted as many serial lines as the VME-based system; therefore, we looked for a PC/104 board with eight ports which we could stack. Unfortunately, the choice of boards with more than 4 ports is limited. We finally chose the Parvus Octal Serial card.

This board is a so-called “dumb multiport serial” board, based on two 16554 UART (which are compatible with the standard 16650 UART) plus a programmable gate array for the address and UART register mapping on the ISA bus. As this board is quite new, we encountered some problems with the on-board gate array (bugs and limitations of programming) but we also found

workarounds. To integrate this board in our Linux TacoBox, we used the standard serial line driver of the kernel (2.0.35). It implements IRQ sharing; thus, we consume only one IRQ for the eight ports of a Parvus board.

Before accessing the extended serial lines through the system device descriptors **/dev/ttyS**, we have several setup steps. All are processed from a single homemade script. Therefore, to install one (or more) Parvus boards, we need to add a single line to the booting scripts. We added a call to our script in the /etc/rc.d/serial file.

The first step is to set up the Parvus card gate array. We wrote a C program to access the gate array registers. The script picks up all the parameters needed for the call to this program and for configuring the serial lines from an easy-to-understand and modify ASCII file. Then the device descriptors /dev/cuaxx and /dev/ttySxx are created. Since we use the standard driver, the major numbers are the same as the ones used for the CPU board serial lines (4 for ttySxx and 5 for cuaxx). As with any other dumb multiport card, the minor numbers start from 64. At this step, we also change the access permissions to the device descriptors to allow any user to use the serial lines.

The last step is configuring the serial ports using the helpful tool **setserial**. We set up each port of a board with a different address but with the same IRQ; for example, addresses 0x100, 0x108 and IRQ 12 for one board and addresses 0x140, 0x148 and IRQ 9 for the other board. At this point, the serial ports are fully integrated in the operating system and can be used from any program.

Figure 4. Manuel (aka RastaMan) after a heavy day of debugging the serial line TacoBox, feeling like he has serial lines coming out of his head.

For our embedded controller, we ported the existing serial-line TACO device server from our commercial OS to Linux. The new device server uses POSIX calls to interface with the serial-line driver and should therefore work with any

OS supporting POSIX. The compatibility of the two device servers and the use of TACO allows us to interchange a VME-based system with a PC/104-based system without changing the client application.

Stepper Motor Controller

The experiments conducted at the ESRF beamlines require precise positioning of many motors to move goniometers, slits, translation stages, etc. Typically, a beamline has more than a hundred stepper motors to align the beamline and move the sample during the experiment (see Figure 5). Motors can be divided into two types: those which can be moved independently and those which have to be moved in sync with the data-taking process. For the independent motors, a TacoBox based on an embedded PC/104-based controller board is ideal—it is compact and can be installed close to the hardware. For the synchronized movements, VME is the preferred solution. It allows us to synchronize the motor movements with the data acquisition via the VME bus without requiring any extra cables.

Figure 5. An example of motor positioning on a beamline—a 6-legged hexapod used to position the crystal in the beam to select only a single wavelength of the beam.

Figure 5

In order to kill two birds with one stone, we chose a stepper motor controller which exists in PC/104 and VME format—the PC68 and VME58 from Oregon Micro Systems. These cards supports step rates of up to 1MHz, very useful for microstepping where the steps are divided by a factor of 1000 to ensure precise positioning. Both cards implement the same ASCII-based controller language. They differ in their register mappings on the bus (one is Intel I/O port-based and the other is m68k memory mapped). The differences are implemented (and hidden) at the level of the device driver. This ensures the TACO device server is identical for both cards. We were fortunate enough to get outside help from Richard Hirst (again) to write the device driver, which is based on an initial version we found on the OMS web site written by Tony Denault of the Institute of Astronomy, Hawaii. The driver implements **ioctl** calls to read position and status and to pass commands to the board. A multi-threaded device server which supports events was developed at the ESRF (based on an initial version by Lucile Roussier of Lure Laboratory in Paris). The server uses POSIX threads on Linux/m68k and Linux/x86. All the software (device driver and server) is available under the GPL from our FTP site at ftp://ftp.esrf.fr/pub/cs/ess/linux/drivers/oms/

Figure 6. Paolo and Andy in their bug-fighting gear with the stepper motor TacoBox (second shelf from the top on the right-hand side) in its rack on the

ID27 beamline clean room. The other boxes are the stepper motor power drivers and the TacoBox power supply.

Figure 6

The PC/104 controller consists of a JumpTEC 486 CPU board and the OMS PC68 controller stacked together (see Figure 6). The TacoBox boots Linux from flashdisk, loads the device driver module, creates as many device descriptors as needed (using major number 26), then starts the device server. On VME, an extra step is needed to program the Vmechip2 according to the ESRF addressing standard.

Once the device server is running, the client requests the server to move the motors. A minor problem we had was ensuring that the PC68 card did not clash with any I/O address already in use, e.g., the address of the network card. This is easy to determine by listing (using **cat**) the /proc/ioports file and choosing one that is not attributed. A more serious problem was with the VME version of the card and the way it handles interrupts when hitting a limit switch. In the end, we abandoned using interrupts with limits; we simply read the status register to find out if a limit had been hit.

For the stepper motor-based TacoBox, the usual PC/104 cabling problems are reduced somewhat because there is only a single flat cable for the motor controller and an Ethernet cable to connect. The entire box cost us approximately 2000 euros for four stepper motor channels and 3000 euros for eight channels (using the PC68 extension board). In our application, we use the PC68 as a simple stepper motor controller; we haven't tried the other features yet (e.g., support for relative encoders, dc motors and servo loops).

Collaborating On Device Drivers

Today, the main drawback of Linux for our applications is still the lack of device drivers for many kinds of I/O boards. PC boards are better supported than VME, but still lag far behind Windows drivers. Our aim is to install as many Linux-based controllers as possible because of their better stability, ease of programming, flexibility, lower cost, etc. In order to achieve this, we need drivers, drivers and yet more drivers.

We are only a small team working on device drivers and are therefore very interested in collaborating with other programmers on device drivers for all kinds of boards. All our drivers are developed under GPL and made available to the external world on our FTP site.

We would be interested in hearing from anyone who has written a device driver for an I/O board for PC/104 or VME. Please send e-mail to one of the authors to

add your driver or name to our database, which we will make available on our web site.

Conclusion

We hope this article has given you a taste of how to build embedded controllers using Linux and how Linux is being used in a research institute like the ESRF. All our software is available free of charge (and guarantee), with source code. At present, we have built only a few embedded controllers based on Linux, but we plan to build many more in the future. Should any real-time problems crop up, we will explore them using RTLinux.

Linux has proven to be a great OS for collaboration. All of our software comes from the Internet, and we have often had direct contact with the authors. We hope to collaborate with more device driver programmers in the future to bring the list of Linux device drivers up to be at least as long as for Windows, if not longer.

Figure 7. Is it Richard Stallman? No—it's Paolo preaching the gospel of Linux and Open Source in the clean room.

Figure 7

References

Acknowledgements

Andy Götz aka Mr. Linux (goetz@esrf.fr) has worked at the ESRF for ten years. His main interests are distributed control, astronomy, travel and Linux.

Petri Mäkijärvi aka Mr. TacoBox (petri@esrf.fr) has worked at the ESRF for almost as long. He is responsible for embedded systems, real time, making great web pages and recently, Beowulf clusters.

Bernard Regad aka Mr. VmeBoot (regad@esrf.fr) has been with the ESRF for six years. His main interests are configuring and booting diskless VMEs, installing Beowulf clusters and 70's music.

Manuel Perez aka Mr. RasterMan (perez@esrf.fr) is a longtime ESRF collaborator and now colleague. His main interests are building the Serial Line TacoBox, programming, cinema and Linux.

Paolo Mangiagalli aka Mr. Medea (mangiaga@esrf.fr) is the newest ESRF member. His main interests are making the MEDEA beamline the best, building TacoBoxes real cheap, and moving hexapods.

Running Linux on a Laptop

Erik Max Francis

Issue #66, October 1999

A quick look at what to look for in a laptop for Linux and how to set it up.

A laptop which runs Linux: most Linux enthusiasts think this is a great idea, but not surprisingly, some people who don't use Linux wonder why you would want to run a flavor of UNIX on a laptop. The answer is simple. If Linux is the operating system you use, the one on which all the applications and software you employ on a daily basis run, then you want it for your portable machine as well.

This article is intended for those who are considering getting a laptop on which to run Linux. It is a quick summary of what you should be thinking about when looking for a laptop for Linux, and a brief outline of what you need to get Linux configured and running on a laptop.

For more information, the reader is directed to the web pages in the Resources section at the end of this article. They will be invaluable in researching and later configuring your laptop for Linux.

Hardware Considerations

The first thing to ask yourself is, how do you want to use the laptop? Is it just for fun? Is it for transferring files between two locations? Is it to be used for word processing, programming or graphic manipulation? Do you need only a simple terminal with which to log in to other machines? Just as with a desktop machine, considerations like these will affect the decision regarding which kind of laptop you should get.

Other questions to consider are: Do you want a CD-ROM drive? Do you want sound, and if so, what quality? Do you want a built-in modem? A built-in Ethernet adapter or PCMCIA, a PCMCIA modem, or neither? Do you want to run X? If so, what video mode do you need (640x480 is the low end, 800x600 is

common, and 1024x728 is approaching the high-end level of the market)? Do you want color, and if so, what bit depth (8, 16, 32 bits)?

Finally, you need to make sure the features you require in the laptop you are considering are supported with Linux. This is far less problematic than it was in the past. When I first installed Linux on a laptop (a Twinhead SubNote with only 4MB of memory), I was taking a gamble that the hardware I needed would be supported; fortunately, it was.

These days, laptop hardware is far less proprietary and you have a much better chance of just buying a laptop and getting it to work. Buying a used laptop with a PCMCIA slot for an Ethernet card is probably sufficient if you just want a machine to use as a terminal without X. If you are planning on doing more with it, you will need to verify that your hardware is supported by Linux. (See Resources.)

Since laptop LCD (liquid crystal display) screens have fixed pixel video modes, they usually emulate lower video modes by duplicating the contents of rows and columns at increments of a set number of rows and columns, respectively. This means that if you get a laptop with 800x600 resolution and you plan to use it just for virtual consoles, you will find that when it is emulating the 640x480 standard VGA mode, the characters will be blocky and ugly. If fidelity is important for virtual consoles (especially if that is all you plan to use), you will want to get either a laptop with 640x480 or one with a considerably higher video mode (1024x768). For these, the virtual consoles will look less blocky and will be easier to read. Some high-end laptops (such as the Sony Vaio PCG-505FX) have configuration settings that don't do this “zooming”, which makes everything easy to read, albeit smaller.

Installation

Obviously, you need a boot disk appropriate to your laptop's hardware, just like any other Linux installation. Laptops generally use IDE, so normally you don't have to worry about SCSI support unless you have special needs (and have a SCSI PCMCIA card).

CD-ROM

Many laptops these days come with CD-ROM drives, making installation easy. The CD-ROM drives are generally IDE interface-compatible, so a standard IDE boot disk will usually suffice. For slightly-incompatible drives, you will need to enter special parameters into the kernel.

Some laptops, such as my Sony Vaio PCG-505FX, have CD-ROM drives available only through PCMCIA interfaces. Surprisingly, these do work, although you

might have to send parameters to the kernel at boot time. For instance, my Sony drive works fine only if I send the kernel parameter **ide2=0x180,0x386** on boot. I could also put an **append** statement in my lilo.conf file.

PCMCIA Support

A large number of PCMCIA devices are supported by Linux: modem cards, network cards, SCSI cards, combination cards and so on.

PCMCIA isn't supported in the kernel, so you will need to use a PCMCIA root disk which starts the PCMCIA daemon, **cardmgr**. Many PCMCIA devices are automatically supported; just insert the card, start the machine with the PCMCIA root disk and away you go. For devices which aren't immediately supported, you will have to do some tweaking of the configuration files (see Resources).

The Brute-Force Method

For older laptops, you might have no option other than the brute-force method. My Twinhead SubNote, for instance, didn't have a CD-ROM drive (almost no laptops had CD-ROM drives back then), and I didn't bother buying the optional floppy drive for it. I was left with one option: buy a 2.5-inch drive adapter for a normal IDE controller, pop open the laptop, take out the drive, plug it into another desktop machine, then install Linux. It is crude, but effective.

Networking

You'll most likely want to have Ethernet support on your laptop. Modem detection and configuration is fairly straightforward on laptops (often completely automatic), since many have built-in modems that respond normally on the serial ports to which they're assigned. Also, most of the standard PCMCIA modem cards are easily supported, even if they are on combo cards with Ethernet adapters, such as the IBM Home & Away card.

Ethernet Configuration

For laptops which have on-board Ethernet (such as the Hitachi VisionBook Pro 7000 series), you use the same method as for a desktop machine to get TCP/IP networking up and running.

For PCMCIA Ethernet adapters, you will need to edit the PCMCIA network options file, probably /etc/pcmcia/network.opts. Editing /etc/rc.d/rc.inet1 or /etc/resolv.conf won't help, because the kernel Ethernet services won't be used.

The format of /etc/pcmcia/network.opts is straightforward; it contains the same sort of options as in /etc/rc.d/rc.inet1, as well as **DNS_1** through **DNS_3**, for specifying domain name servers, and **MOUNTS**, for specifying NFS mounts which also must be listed in the /etc/fstab file. It also contains other options for configuring your Ethernet card in the event it is not automatically supported. (See Resources.)

Switching Between ISPs

If you are like me, you have an Ethernet network at home and at the other places you take your laptop (a friend's house, work, etc.). I've found that generally I have no need to use the laptop “in transit”. Since it is possible that I may not get around to plugging it into an AC adapter wherever I'm going, I don't need to keep the machine on while I'm switching from one network to another.

If this is also your situation, there is a simple solution. When switching networks, all that must be changed is either the /etc/rc.d/rc.inet1 and /etc/resolv.conf files, in the case of built-in Ethernet support in your laptop, or just /etc/pcmcia/network.opts if you are using a PCMCIA Ethernet card.

Listing 1.

Let's say you want to call the two locations (though there can be more) between which you want to transfer “home” and “away”. Rename the above-mentioned files to have extensions of .home and .away, for the configuration of each location respectively, then use a simple shell script such as the one shown in Listing 1 to point to the proper location. This script makes symbolic links for each configuration file, pointing to the corresponding location-specific file, if it exists. If it doesn't, it won't do anything, so nothing will break.

When you are ready to shut down your laptop for the journey, log in as root, run this program with a single command-line argument which is the location to which you are moving (e.g., away), then shut it down. When you start it back up, it will be configured for its new temporary home.

X Window System

Configuring X is fairly straightforward for laptops with supported hardware. The only place to be cautious here is making sure you have a laptop that supports Linux with the display hardware on the laptop (which is still somewhat proprietary, but far less so than when laptops first began coming out), has sufficient memory to run the X server, and runs the video mode and bit depth you need. The newer, high-end laptops often use the NeoMagic chip set, which

is fully supported only in XFree86 3.3.2 and higher, so you may have to upgrade.

The best resource for determining whether a laptop will meet your needs is the *Linux on Laptops* web page (see Resources).

> Erik Max Francis is a UNIX engineer who lives in San Jose, California. His main interests are programming, Linux, physics and mathematics. He has been using Linux exclusively at home since kernel version 1.2.8 and has been reading and contributing avidly to Usenet since 1989. He can be reached via e-mail at max@alcyone.com.

Resources

Archive Index Issue Table of Contents

Advanced search

Transvirtual Adopts Microsoft Java Extensions

Craig Knudsen

Issue #66, October 1999

Mr. Knudsen tells us why this company chose to add MS extensions to Kaffe, the open-source Java implementation.

Kaffe, first released in 1996, was the original open-source Java implementation. Initially developed as part of another project, it grew so popular that developers Tim Wilkinson and Peter Mehlitz founded Transvirtual Technologies, Inc. with Kaffe as the company's flagship product. In July of 1998, Transvirtual released Kaffe OpenVM under a GNU license.

Kaffe's multi-platform support and small footprint have attracted many users. On-line magazine JavaWorld gave Kaffe their Editors' Choice Award for Best Virtual Machine for 1998, stating, “Java has been waiting for Kaffe for a long time. A proper open-source, cross-platform JVM, free of restrictive licenses, is a must for the complete acceptance of Java.” Kaffe was originally released as source code long before the term “open source” had been coined. It was the first Java Virtual Machine (JVM) for many operating systems and was also one of the first to offer a Just-In-Time (JIT) compiler for many architectures, improving Java's runtime performance.

Microsoft's Java extensions have created a good deal of controversy. Sun claims that Microsoft violated the terms of its Java licensing agreement and has filed a highly publicized lawsuit. Microsoft's version of Java was not “write once, run anywhere” as Sun intended Java to be. To much of the Java community, it appeared as though Microsoft was attempting to splinter Java with extensions like J/Direct and new Java keywords (“delegate” and “multicast”) that gave Java developers new capabilities not otherwise available in Java. Applications using these extensions are required to use Microsoft's JVM.

At first look, it may seem surprising that a company known for its open source, clean-room implementation of Java would add support for Microsoft extensions to the language. However, by adding support for these extensions to Kaffe,

Transvirtual is allowing Java programs developed under Windows (using Microsoft Visual J++) that make use of the extensions to run on non-Windows platforms such as Linux, FreeBSD and Solaris. Additionally, Transvirtual is providing a new source for Java on the Windows platforms.

The first extension Kaffe supports is delegates. The delegate keyword essentially provides a function pointer. This allows an application developer to tie an event to a specific user interface element (such as a push button). Microsoft added the new “delegate” keyword to the Java language to support this. (At the time delegates were introduced, Java was at level 1.0.2 and had not yet introduced the new 1.1 event model which provides similar capabilities.) Microsoft Visual J++ is required in order to compile the Java code if you make use of the delegate keyword. Up until now, you also had to use Microsoft's JVM to run the compiled Java application. Kaffe's support for this extension allows non-Windows users to run Java code that makes use of the delegate extension. For now, you will still need to compile your application with Visual J++. But, you can deploy the compiled Java code on Linux or any other Kaffe-supported platform.

Delegates is only the first extension that Kaffe will support. Microsoft's J/Direct provides similar functionality to Sun's JNI, giving developers access to application-specific functions. Why would anyone choose J/Direct over JNI? According to Transvirtual founder and CEO Tim Wilkinson, “I wouldn't primarily expect UNIX users to pick these extensions up and run with them, but since Windows users are using them, I don't want them marooned on Windows-NT for the rest of their natural lives. J/Direct support, in particular, makes it far simpler to move Java code which uses an application-specific native library to UNIX.” Wilkinson went on to say that J/Direct “provides similar functionality to JNI in, as far as I'm concerned, a much cleaner fashion.”

Although funding was supplied by Microsoft to add these extensions to Kaffe, the relationship was initiated by Transvirtual. Transvirtual wanted to give users the freedom to choose a non-Microsoft platform even if they developed their code with Microsoft tools. Microsoft made documentation available on their web site (see Resources), but did not contribute any code or other intellectual property. One of the conditions of Transvirtual's support for the extensions was that the result would be released as open source under the GNU GPL.

Why would Microsoft be interested in open-source implementations of their Java extensions? Sun's lawsuit against Microsoft has placed the future of Microsoft's Java implementation in jeopardy. The extensions added to Kaffe are at the heart of the lawsuit. Many companies are beginning to look for a second source for Java. A version of Kaffe for Windows with AWT 1.1 and native threads support was scheduled to be released in late July.

By supporting delegates (and eventually J/Direct and Java/COM), Transvirtual has made it possible to take code developed using J++ and deliver it to Linux. One of the great strengths of Linux is its interoperability with other operating systems. By providing this technology to the Linux community, Transvirtual has added to this strength.

Resources

Kaffe OpenVM's Supported Platforms

Craig Knudsen (cknudsen@radix.net) lives in Fairfax, VA and telecommutes full-time as a web engineer for ePresence, Inc. of Red Bank, NJ. Craig has been using Linux for both work and play for three years. When he's not working, he and his wife Kim relax with their two Yorkies, Buster and Baloo.

Archive Index Issue Table of Contents

Advanced search

Houdini: Magic Doesn't Just Happen

Michael J. Hammel

Issue #66, October 1999

Side Effects Software pulls the Linux penguin out of its hat with a port of Houdini.

We all know about those movies with the great special effects: *The Matrix*, *Armageddon*, *The X-Files Movie*, *Godzilla*, *Lost In Space*. The list goes on and on. Up until now, most of those effects have been done on SGI workstations. SGI has been a leader in this field for many years. However, in the past few years, a migration to low-end, low-cost, Intel-based PC hardware has occurred. This migration has included Windows, the only OS considered to work well on those Intel desktop computers. That situation is about to change.

Side Effects Software (http://www.sidefx.com/), a graphics software company founded in 1987 in Toronto, Canada, is porting its only product, Houdini, to Linux. This is big news in the graphics world. Houdini is the software used to create many of the stunning effects in the movies I just mentioned. To those in the graphics arts world, this is the equivalent of Oracle, Corel or Lotus porting to Linux. It gives credibility to the platform in an area where it had been mostly a curiosity.

Linux has had a major role in a few movies in the past. Darryl Strauss' article (*LJ*, February 1998) about Digital Domain's Linux render farm for *Titanic* proved that, but Linux hasn't been used as the primary graphic artist's platform for these movies. Like so many other industries, the film and video industries have

been using Linux as a server, a box off in the corner happily crunching numbers or dishing out files over high-speed networks. Now, Linux gets to take center stage on the desktop.

Screen Shot 1.

Side Effects' Houdini product is a complete 3-D solution, encompassing modeling, compositing, lighting, particle systems, texture management, rendering and animation features. Currently in its third release, Houdini was one of the first modeling and animation products to adopt a procedural approach to 3-D. Recently, Houdini received an Academy Award for “Technical Achievement”, presented to four of the company's original developers. Used by companies like Digital Domain, Blue Sky/VIFX Studios and Centropolis Effects, Houdini is a heavyweight in the film industry.

Previously, Houdini has been supported only on SGI Irix workstation class systems. They recently announced support for Windows NT. The port to Linux comes at a time when many hardware vendors are looking for reasons to bring their high-end graphics cards into the Linux fold. Paul Salvini, Director of R&D at Side Effects Software, talks of it as the “chicken and egg” problem:

> Doing a product like this for Linux required hardware acceleration to make it really viable, but hardware acceleration often requires applications in order to warrant drivers to be written. From a graphics workstation point of view, Linux isn't ready. There haven't been that many proper drivers for hardware acceleration for OpenGL under Linux. There are a number under development, but as far as state of the art, there aren't that many in production. The reason is that there aren't any applications pushing the need for these drivers. The applications want the drivers; the drivers are looking for applications.

Linux is a viable, popular, rendering platform. Many houses that formerly used SGI servers for rendering are moving to, or at least considering, low-cost Linux platforms for their rendering farms. This might be due to their familiarity with UNIX or concerns with Windows NT stability or even if they just prefer not to have a mixed UNIX/NT environment.

Upending Traditional Linux Development Models

With Linux, most of the current software development is driven from the bottom: what happens in the kernel drives what happens with applications. Side Effects thinks, for the high-end graphics market at least, this needs to be a bit different. Houdini is an application, with low-level needs that aren't quite available the way Side Effects wants them. Salvini said:

> What will happen is the missing pieces underneath will just naturally fill in—development is being driven from the top of the software chain. This is an exciting change in the way development happens for Linux.

Motivations

Side Effects wanted to do their part to move things forward by providing the application. By providing a Linux solution, they felt they could provide their customers with the best of both worlds, low-cost Intel-based platforms and the familiar UNIX environment. Plus, having the kernel source offers the effects houses, which tend to have very technical needs, the opportunity to make modifications they might not otherwise be able to perform.

Screen Shot 2.

Part of the reason they considered doing the port was the interest shown by a number of effects houses for a Linux-based Houdini. The interest was “provisional”, considering the lack of workstation-class hardware support (i.e., the OpenGL hardware acceleration), but Side Effects took these requests to heart and felt they could help move the process along by providing the application side of the equation.

Although they didn't have any actual customers signed up for the port—just customers showing interest in its possibility—and even though no satisfactory 3-D hardware acceleration was available, Side Effects felt they had to move forward with the project. Mr. Salvini told me:

> We did this as much because we felt it was the right thing to do as anything else. To sit around complaining that no one has developed proper hardware acceleration is kind of hypocritical if you're not providing the application that will take advantage of it. You've got to put your money where your mouth is and do the work.

The Port

Before starting work, Salvini said Side Effects spoke to Darryl Strauss at Digital Domain, Xi Graphics and Metro Link about the hardware support that might be forthcoming from the X server vendors. Initially, they started with XFree86, then moved to Xi's AcceleratedX to make use of support for overlay planes, something neither XFree86 nor Metro Link supported at the time. The overlay planes allowed them to speed displays by not having to continually redraw sections of the screen. The dependency on Xi's server is temporary, however, since both the other vendors are working on overlay planes, plus Houdini doesn't have a huge need for them. In fact, Side Effects may provide Houdini in

a version that works without overlay planes. The drawback will be a hit in performance, but the program will still be fully compatible with the version supporting overlay planes.

Salvini couldn't talk about which 3-D cards they were looking at, due to non-disclosure agreements. He did say that on the NT port they had worked with IBM and Intergraph, 3-D Labs, Diamond Multimedia and Accel Graphics. Intergraph provides their Intense3D, also known as the Wildcat 4000 card, for the IBM Intellistation. He said response from all these vendors was very good for that port. Since all of these card manufacturers have shown varying interest in the Linux environment, one can only imagine what, if any, their involvement with upcoming hardware announcements for Linux might be.

Beyond hardware acceleration, other issues which the porting team had to deal with are easily recognizable to anyone who has done UNIX-to-UNIX porting in the past: sockets, shared memory work, clone vs. fork/vfork. These are the typical issues encountered when porting applications from IRIX to another UNIX platform (or vice versa). Linux, for all practical purposes, fits into the UNIX marketplace quite well in this respect.

As for doing the actual work, Side Effects was able to make use of all the tools they already had. Salvini told me:

> The good thing about Linux was that the tools we did need—for debugging and memory checking, for example—were available for free and we already had a good idea what was going on there.

Even so, it apparently took quite a while to get all the graphics pieces working together, since they “were always downloading the latest bit of this or that.”

Salvini wouldn't term it “an easy port” from the SGI version, due to some tricky things they do concerning performance issues, but he did say it took two engineers about five weeks to do the work: “less time than moving to NT, but it was still longer than we had hoped.”

The port was actually done as a fun thing at first, and they'd figured it might be something they could finish in a weekend. We both laughed when I pointed out that, even with Linux, such a thing as a “weekend port” was still a bit of a Holy Grail for most packages.

Ready to Rumble

At this point the main graphics work has been completed, but audio and MIDI support has not been done. MIDI is a fairly small portion of the product, but

they'd still like to finish it sometime soon. Unfortunately, UNIX audio interfaces are fairly non-standard. The developers are looking at the Open Sound System as a possible way to standardize audio ports, but no decision in that arena has yet been made.

The Linux port is actually ready to deliver, minus the small audio issues, but they're waiting on a proper license manager. Houdini will go to beta when that tool is ready. Due to the bleeding-edge nature of this project, they don't feel they can use any of the available license managers, so they may go with one built in-house.

The lack of hardware acceleration on the Linux box means it does not compare to the IRIX boxes for performance. In terms of everything else—rendering, simple shading, wire frame preview—Linux stacks up fairly well. Salvini said:

> It's surprisingly good without that one piece. I can't wait for that final chapter to close, for then it will be an amazing platform.

Houdini pushes OpenGL pretty heavily, and as Paul pointed out:

> We'll be a pretty good test of OpenGL for vendors on Linux.

Although used primarily for film, Houdini is also used by a few game companies in Japan. That is one region Side Effects is pushing for better Linux support. They are also involved in some scientific visualization and even a bit of industrial design, both areas that currently are seriously under-represented in the Linux marketplace.

Many special effects shops use multiple packages for their work, one for modeling, another for animation, another for effects and another for compositing. Ultimately, you'd want to see all of those tools for Linux. The toughest one will be something like 3D Studio Max, because it has been designed specifically for the Windows platform, unlike many other tools that have their roots in the UNIX world. However, tools like 3D Studio Max are important for the graphic arts world on Linux because they're mid-range tools, where many people get their start. SoftImage and Alias/Wavefront will be easier ports, because they are used to the UNIX and X environment. SoftImage is owned by Avid, but neither the SoftImage nor the Alias products have been talked about lately for the Linux platform.

Many hardware vendors are aware of Side Effects' port, but Paul is under non-disclosure with those vendors and couldn't talk about their plans. Most of the software vendors who have talked to Side Effects have asked them about the

port—they want to know what Side Effects knows that they don't. Most of this has to do with questions relating to hardware acceleration. That's the key for which many application vendors are waiting. Like the public, those other vendors will have to wait until the workstation, graphics card and X server vendors start to make their own announcements.

Side Effects Take On Open Source

The developers and management at Side Effects have talked about open source quite a bit, says Salvini:

> There was a great article on it by someone at Netscape about the business case for Open Source and how you might make money with that.

They had never had Open Source, but this got them tossing the idea around a bit, albeit without releasing all the bits and pieces of Houdini. The concept raised a lot of "what if" questions, but so far it hasn't gotten much past that point. They are obviously intrigued with the idea, however.

Side Effects does offer source access in a traditional sense with a developer's kit allowing access to header files and libraries, but the core remains proprietary, partly due to their concerns about maintaining product consistency. They do believe they offer extensibility with the plug-in interface. Salvini says that with high-level applications like Houdini, access to the core code is not as vital as access to the operating system source.

Paul doesn't think anyone at Side Effects actually uses Linux at home, although there may be one guy who has it. In fact, he says most of their developers don't even have systems at home. Apparently, they have lives when they leave work. I had to wonder a bit what that might be like.

Now that the Penguin is Out of the Hat

Although the Houdini port to Linux is big news for graphics nuts like me, don't expect to run out and pick up a copy of Houdini off the shelf at CompUSA. A single-seat license for Houdini runs about $17,000 US, with the developer's kit running an additional $4000 US. This is very high-end software for professional organizations.

Considering this high price, I asked Paul about his take on a more affordable modeling package that has developed a big following—Blender. He replied:

> You can't compare the two. They really aren't in the same class of product. I didn't play with [Blender] long enough to do it justice, but obviously its price point is

> such that people will be able to take advantage of it just from a personal development level, i.e., an experiment in playing with graphics tools. You don't do that with a $17,000 product.

Unlike Blender, Houdini is a product that needs a bit of serious commitment from the user in order to warrant its expense.

However, the news of the Houdini port is important more for what it represents, the credibility it gives to Linux in the desktop graphics marketplace, than the product being produced. Salvini was right when he told me,

> This article won't be about trying to sell Houdini to the readers—it's not priced for the average user—it's about how Linux is developing in the high-end, graphics workstation market. Side Effects has been pushing the hardware vendors in this direction quite a bit and I feel we'll be seeing some great success in the next year, to bring Linux into the graphics workstation market, which it really hasn't been to date.

He's right. Now that Houdini has made the jump, things truly are about to change.

Michael J. Hammel (mjhammel@graphics-muse.org) is a graphic artist wanna-be, a writer and a software developer. He wanders the planet aimlessly in search of adventure, quiet beaches and an escape from the computers that dominate his life.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

UpFRONT

Doc Searls

Issue #66, October 1999

Strictly On-Line, *LJ* Index and more.

Strictly On-Line

Open Source Software for Real-Time Solutions by Charles Curley compares Cygnus's eCOS to RTLinux, both designed for the embedded system market. Is there a place for both?

Web Client Programming Using Perl by Robb Hill describes how to monitor your own web site using Linux and Perl, in particular the LWP modules. He supplies scripts for creating an HTTP ping utility, paging using the Skytel web site and paging for SNPP servers. Use these scripts and keep cooking with Linux.

Java Servlets by Doug Welzel is an introduction to writing and running Java servlets on Linux. Servlets are Java programs which extend the functionality of the server and can be used to replace CGI scripts.

Bisel Bank by Pablo Trincavelli is a “Linux Means Business” article describing how a bank in Argentina uses Linux for testing database and web applications.

Perl in a Nutshell by Jan Rooijackers is a book review describing the contents, what is good and bad about it and why you might want to buy it. Perl is one of the most popular scripting languages in use today and the “nutshell” books are one of O'Reilly's most popular series. Don't miss this one.

Java 2 Software Development Kit by Harry J. Foxwell is all about the latest version of the Java SDK from Sun. Mr. Foxwell is a System Engineer for Sun and knows his subject well.

LIMP: Large Image Manipulation Project by Valient Gough tells us about designing a new library for processing large images using a minimal amount of memory. In this project, he uses C++, the Qt library and plug-in types for a

number of interfaces. LIMP is being used for scientific image-processing needs, particularly aerial and satellite images.

Games Focus—>Dungeon Crawling

Some rogues are better off in dungeons. If you too are better off in a dungeon (or if your friends just think so), have a look at some of these classic console character-based role-playing games available for Linux. Have you ever wanted to be an @? Well, now you can, and as an @, you can guide yourself through hundreds of layers of algorithmically generated dungeons, encountering crazy creatures, mysterious treasures, problematic nymphs and even the notorious Kobolds, who burst when you strike them! What ever *am* I talking about?

Screen Shot

Rogue was the creature that started it all. Written nearly 20 years ago, it was designed to run on “dumb terminals”, machines which were connected to mainframes but had no special powers of their own (such as graphics, for example). The intent was to produce a character-based adventure game, using the curses library, which would produce a new adventure every single time instead of reiterating the same plot over and over. This approach worked and produced a game which could surprise even its creators. Thus began a new genre of computer game, and generations of dungeon-crawlers were spawned.

Hack was one of the first Rogue-like games, and it introduced a new component—pets. You got to have a dog or cat wander about with you in the dungeon, and this animal was good company. Many items and new features were added, and the game became popularized in various formats across several platforms. As a child, I heard stories of a mysterious game called Hack which was supposedly a miraculous, ingenious game, vast and complex, like nothing I had seen before. In those days, stories soared into legends (like the rumor that Bard's Tale IV had been created but required a Cray supercomputer) and when I finally got around to Hack on the Amiga, it was already in another incarnation.

Screen Shot

NetHack, the direct descendant of Hack, is the most famous of the Rogue-inspired games. It is extraordinarily complex, offering all sorts of classes, weapons, scrolls, magical rings, potions, creatures, locations and plots. The idea is that once you have outspent yourself (and your parents), you're better off seeking your fortunes underground by retrieving the Amulet of Yendor (well, so says http://www.gnu.org/; the game text has a different interpretation). Hence, you can become a Valkyrie (or a Wizard, Samurai, Rogue, Priest, Knight, Healer, Elf, Caveman, Barbarian, Archeologist or Tourist) and go out questing. There are

dozens of levels which become immensely complicated, and the game draws on strategic thinking, cleverness and long-term strategy. NetHack is now available in two graphical versions, one based on Qt (QtNetHack) and one based on gtk (GnomeHack). These graphics are excellent and I recommend taking a look. This is a deep game (smile) which takes some time to get used to, but it's good fun and since it *is* a classic, it is a good way to expose yourself to hacker culture.

Screen Shot

Moria is a Tolkien-inspired descendant of Rogue, written in 1983 for VAX machines and ported to UNIX in 1987. The point is to kill the Balrog. Dungeon levels are quite large, taking up several screens, as opposed to the single-screen dungeon levels found elsewhere. Hence, it is quite a bit larger than the original Rogue. Also, it features a town level, in case you want to come up for air. You can choose from numerous races and classes—if, for example, you've always wished to be a Hobbit (or believed you were one), now's your chance.

Angband is another Tolkien-inspired game (well, what isn't?) which was derived from Moria in 1990. The idea is to descend into a very deep dungeon and kill Morgoth, the Dark Enemy of Middle-Earth. The atmosphere is more serious than that of NetHack, so if the persistent humor and silliness of NetHack end up spoiling the fun for you, Angband is a good alternative. A developmental, multi-player version is also available.

ADOM, Ancient Domains of Mystery, is yet one more Rogue-like game which differs a bit from the others and offers much more in some areas. It has many different character classes, and the magical characters are especially interesting. A commercial version is planned, as well as a real paper-and-pen RPG (role playing game). ADOM is still version 0.9.9, but seems to have a large following especially among teenagers and players under age 10. It is being actively developed—the author doesn't seem to have run out of energy so far.

CrossFire is a different kind of game from the rest of these dungeon crawlers. The Linux Game Tome (http://happypenguin.org/) describes it as a cross between NetHack and Gauntlet, and that's actually fairly accurate. The game is graphical, multi-player (!) and immense. With over 150 different monsters, about 3000 maps, 19 character classes, about 65 different weapons, dozens of armours, helmets, shields and clothings, and 18 levels of magic available to wizards (with roughly 85 spells at last count), CrossFire is a whole different world in which you and your friends can live. Any number of people can have clients (even available for Java and Win32), but the server has to run on a UNIX-based system such as Linux. If you're tired of being an @ and want to be an animated graphic again, here's where you can do it!

—Jason Kroll

LJ INDEX—October, 1999

- Year Alan Turing wrote *Computing Machinery and Intelligence*: **1950**
- Turing estimated the binary digit storage capacity of the human brain to be: **1010**
- Number of years Turing estimated would pass before computer storage would reach 109: **50**
- Wholesale price of a 109 (1GB) hard drive in August 1999: **$150** US
- Value of the Loebner Prize for the first computer to pass the Turing Test for machine intelligence (i.e., a computer in which the responses to the test are indistinguishable from a human's): **$100,000** US
- Year the Loebner Prize was created: **1990**
- Percent chance given by Turing in 1950 that a computer would pass his test by the year 2000: **70**
- Number of computers thus far to win the big prize: **0**
- Number of correct answers given by "Ask Jeeves" on July 27, 1999 to the question "Who is Linus Torvalds?" : **0**
- Year Ask Jeeves, Inc. was founded: **1996**
- Revenue of Ask Jeeves in 1998: **$450,000** US
- Market capitalization of Ask Jeeves at 4PM on July 2, 1999, at the end of its first day as a publicly traded company: **$1.7 billion** US
- Total exports of the Congo in 1998: **$1.2 billion** US
- National budget of Paraguay in 1998: **$1.2 billion** US
- Millions of desktop systems at the end of 1998: **89**
- Millions of desktop systems at the end of 1997: **79**
- Windows 95 operating system market-share percentage: **57.4**
- Windows 98 operating system market-share percentage: **17.2**
- Windows NT operating system market-share percentage: **11**
- MacOS operating system market-share percentage: **5**
- Linux operating system market-share percentage: **2.1**
- Windows 3.11 operating system market-share percentage: **1.1**
- OS/2 operating system market-share percentage: **.5**
- Percent increase in 1998 Linux shipments over 1997: **212**
- Estimated percentage compounded annual growth rate for Linux between 1999 and 2003: **25**
- Estimated millions of Linux customers worldwide: **10**

- Number of e-mails received in August at info@linuxjournal.com, asking the stock symbol for the company "Linux": **14**
- Number of times Tux, or a reasonable facsimile thereof, appears in the August issue of *Linux Journal*: **48**
- Number of years required to build the Biltmore estate in Asheville, NC—the largest house in the U.S.: **6**
- Number of years required to build the Gates family home, referred to as "The House" in Medina, WA: **7**
- Number of square feet at Biltmore Estate: **175,000**
- Number of square feet at "The House": **66,000**
- Number of square feet at The White House: **67,000**

Sources

1. *Computing Machines and Intelligence*, Alan M. Turing, www.msu.edu/user/vattervi/turing/premium.html
2. The Loebner Prize, www.loebner.net/Prizef/loebner-prize.html
3. Red Herring on Ask Jeeves' debut, www.redherring.com/insider/1999/0702/inv-askjeeves.html
4. AskJeeves.com, http://www.askjeeves.com/
5. International Data Corporation (IDC), http://www.idc.com/
6. Jason Schumaker, *LJ* staff
7. The Biltmore, http://www.biltmore.com/
8. www.usnews.com/usnews/nycu/tech/billgate/gates.htm

Who is Jeeves?

Remember the Turing Test? That was Alan Turing's famous test of computer intelligence, first published in 1950. Essentially, a computer will pass the Turing test for human-grade intelligence when it answers questions in a way that is fully human—that is, indistinguishable from answers given by a human being.

The search-engine business lives entirely outside Turing's shadow. Stupid answers are just not a problem—they're a standard. The more, the better. Case in point: the very popular and well-promoted site, Ask Jeeves (http://www.askjeeves.com/).

Jeeves is a bald and dapper cartoon that insists, like the game Jeopardy, that you frame your search in the form of a question. Here is a recent "dialog" I had with Jeeves:

Doc: Who is Linus Torvalds?

Jeeves: I think you may have misspelled something. Did you mean: Who is Linus Travelogs? Travelled? Traveled? Travailed? Travolta? Trivets? Trifocals? Travelogues? Gravitas? Traffics? Groveled? Grovelled? Gravelled? Graveled?

Doc: Huh?

Jeeves: Where can I read the chapter from Bulfinch's mythology about Orpheus and Euridice, Aristaeus, Amphion, Linus?

Doc: Uh. Let's click on Ask! again.

Jeeves: Linus was the instructor of Hercules in music, but having one day reproved his pupil rather harshly, he roused the anger of Hercules, who struck him with his lyre and killed him.

Doc: Thank you. —Doc Searls

Screen Envy

It used to be flying toasters, tanks of fish and kittens cavorting on black backgrounds. But that was, well, very eighties. The most stylish new screen saver for Windows comes, of course, from Italy—pippo.com, to be exact. It defaults your idle screen to something much more Finnish or Antarctic or Linuxish, anyway. It's called “That Linux Feeling”, and it gives your Windows box that Linux look. To check out the latest, visit http://howto.linuxberg.com/saver/.

Actions Speak Louder

Apparently, the rumors are true. According to a report in ZDNet UK, Adobe's John Warnock recently caught wind of a planned company upgrade to Windows NT and overturned it in favor of Linux, which he describes as “a perfectly viable alternative to NT”. But he also said,

> I want to pay for an operating system from a vendor with a contractual relationship that gives me recourse if things go wrong. Some people who have this utopian view that everything should be free don't understand the necessity for governments or corporations.

Of course, we wouldn't want to include Caldera, Red Hat or Linuxcare among those utopians. They're just corporations which deliver exactly what Mr. Warnock wants.

Who Says?

Who says Linux isn't for desktops? Ask the guys who get the service calls.

In the second quarter of this year, Linuxcare noticed a 27% increase in the number of desktop incidents, while service calls for file, print and web servers went down. Linuxcare co-founder David Sifry told us:

> This industry is really quite interesting. The number of desktop incidents we get has surprised us. Many people are using Linux as desktop workstations for software development, VLSI design, and in financial services, among other things. And we have no idea where it will go next.

Here are the numbers:

	April	May	June
Desktop	52%	59%	66%
File/Print Server	28%	30%	17%
Web Server	20%	11%	17%

Source: Linuxcare

Spending an evening enjoying Niagra Falls while attending COMDEX Canada in Toronto (July 15) are Evan Leibovitch of LPI, Matthew Rice of CLUE, Matthew Cunningham of *Linux Journal*, Dana Epp of CLUE, Mart Withers of Caldera and Allan Smart of Caldera.

Install Fest Tips

1. Find a location that is well-known in your area and has plenty of parking. Be sure it has plenty of space and sufficient power sources. An added plus is having carts available to help people bring in their machines.
2. Host a DemoFest at a meeting about a month ahead of time to generate interest and recruit help.
3. Publicize in the education community (universities and secondary schools) and computer stores, libraries and major corporations (flyers and brochures in the cafeteria or break room). Ask about advertising on local cable access television and community calendars in newspapers.
4. Study the Linux Hardware Compatibility FAQ and the Linux Hardware Incompatibility FAQ and bring copies for your workers.
5. Practice installing Linux on a low-end PC to measure the amount of time it will take to complete. Do the same on a notebook computer and over a network (if you plan to offer this option).
6. Ask companies in the industry for CDs, literature and other items to use as giveaways.
7. Ask attendees to make a reservation in advance, stating the type of PC they will be bringing. Prepare for some to turn up without reservations.
8. Have an area at the front door for signing in and giving away promotional material.
9. Plan on giving all users a Linux CD, so they can distribute Linux to others.
10. Have random giveaways of books, T-shirts, etc.
11. Include Linux training seminars presented in a separate room.
12. Be kind to your workers. Have water and drinks handy. Order in lunch. Make sure they take breaks.
13. Bring a box of extra parts that people no longer need—you may need them to complete an installation.
14. Bring a camera.

—Michael Roberts, Cincinnati GNU/Linux Users Group

WILL YOU BE CRACKED NEXT?

Melissa—Explore.zip—Back Orifice. If you think there has been a bad rash of viruses and crack attacks lately, you're right. And security experts say it's going to get worse, not better; the frequency of crack attacks is rising exponentially. So are the money losses from the problem. Computer Economics, a research firm in Carlsbad, NM, reports that American businesses lost $7.6 billion US due to software viruses during the first half of 1999—more than in all of 1998.

Curiously, the massive mainstream media coverage of these incidents completely fails to mention the one thing they all have in common: Microsoft Windows. Non-Microsoft operating systems such as Linux are invulnerable to macro attacks, immune to viruses, and can laugh at Back Orifice.

This simple fact explains why your Internet service provider never suffers from viruses; essentially all ISPs run their services off UNIX boxes, and about 40% of them run Linux. Evidently, businesses are finding this an increasingly attractive option; a recent Computer Associates survey reports that 49% of information technology managers describe Linux as “important or essential” in their enterprise plans.

One of the reasons for this trend is definitely security. Anyone running a Microsoft operating system on a machine visible from the Internet is just begging to be cracked. If you're concerned with computer security, you need to understand why—and why Microsoft will not and cannot fix the problem.

Linux and other operating systems like it were designed from the ground up to be used by several people on the same machine, and to protect those people from each other. The user interface of Linux is separated from the kernel, the privileged operating system core. And the kernel is carefully protected from being modified by ordinary programs. This is why Linux doesn't get viruses.

Microsoft Windows, on the other hand, has a one-person-per-machine assumption built deeply into it. There is no internal security, and the Windows kernel is not protected against being modified by user programs. In fact, the user interface of Windows is wired right into the kernel. This is why hostile programs coming in over an Internet connection (such as Back Orifice) can reach right through the user interface, deep into the operating system core, and infect it.

If you value your data and your privacy, you need to understand that Microsoft cannot fix this security hole. Too many applications (including Microsoft Office and the IIS web server) actually *depend* on the lack of security in the system.

Furthermore, the fact that the source code for Windows is closed means it never gets properly audited for security problems.

How does Microsoft deal with this? Not well. Mainly, they tell lies and try to confuse the issue.

On August 3, 1999, Microsoft put a machine running a beta of its new Windows 2000 operating system on the 'net and challenged crackers the world over to break into it. A few hours after the announcement, the machine crashed. Microsoft spokespeople subsequently claimed that it had been brought down by electrical storms.

But the machine's own error logs showed there had been nine crashes due to errors in Microsoft's own software, not the weather. Furthermore, crackers did indeed get in and alter a guest book application during the short time the machine was actually up—a fact Microsoft tried to dismiss as irrelevant.

A few hours after Microsoft's challenge was announced, a Linux company in Wisconsin matched it. During the following three days, their Linux machine withstood 6,755 attacks without crashing once.

Which system would *you* rather trust your critical data to?

—Eric S. Raymond

Flattery (Venus or Hedwig?)

Linux Mandrake 6.0 is out. I received it twice in the mail, so I know. I also put it on a partition of a VArStation to check it out. That's when the déjà vu came. I knew I had seen that screen before, “Welcome to Linux Mandrake”. Yes, dozens of times, only I seem to remember the phrase was “Welcome to Red Hat Linux”.

Linux Mandrake, you see, is based on Red Hat. It has a newer kernel, runs a well-configured KDE instead of GNOME, professes to be optimized for Pentium family processors, includes the usual Netscape, StarOffice 5.1 (the new version) and Corel Wordperfect, but at this stage it's still more a derivative of Red Hat than a totally independent distribution, at least so far.

One thing I discovered, though, is that Red Hat *does* successfully probe hardware. In fact, on the VArStation, Mandrake's installer (which is Red Hat's installer with the title changed) probed *everything*, which is a big improvement from E machines where Red Hat found nothing but the mouse. Even my home computer was never Red Hat probe-able, but then again, it's nearly a century old in computer years. The point is that when choosing between a VArStation, an E machine, and a “computer” from 1996, go with a VAr. Still, how do you decide between Mandrake and Red Hat?

Many people insist that Mandrake is better than Red Hat. Mandrake does come out later (by necessity), so it *is* more current. It also runs KDE, which seems to be winning the desktop's favor, and its configuration is pretty nice. It has more menu entries than I recall in Red Hat, and more useful links on the desktop (such as XKill). Mandrake also has some kind of automated online software upgrade function which seems to work—it would be so very Red Hat to have something like that. Mandrake also includes 5 CD-ROMs (though with less software than SuSE) and 100 days of e-mail support. (Well, phone support is a bit too social for us computer types anyway). And, it costs less.

Red Hat, even if it is older by an insurmountable couple of months, does have some points in its favor. For one, it has 700+ pages of documentation compared to Mandrake's concise 189. Also, Red Hat is apparently competing with Linuxcare in terms of support, whereas I don't know how Mandrake performs in this area, though they seem devoted to helping the new user. The Mandrake web site is full of enthusiasm about supporting new users, and maybe Mandrake, with its cute magician logo (Blue Hat?) will take an active role in bringing denizens of other OSes across the mountains to the western paradise that is Linux.

Mandrake is such a mysterious, exotic name, one might expect something a little more intimidating than a distribution for newbies (maybe Linux for sorcerers and witches or something). Still, it's a fine distribution, full of energy, with a following, and it's a bit funny (though maybe not intentionally), if you go for that. Linux Mandrake, based on Red Hat, copied Debian's login penguin, uses a blue hat for its logo, the BeOS color scheme in KDE, and named the distribution Venus (that's almost as good as naming a computer Amiga and

then naming its chips after the developers' girlfriends). I hope Red Hat responds by naming their next distribution Aphrodite; when I make my distribution which has snakes growing out of the monitor, I'm going to name it... Well, in any event, Linux Mandrake does a good job of introducing Linux neophytes to elements of a few distributions. It may be among the better ones out there, especially for beginners.

Stupid Programming Tricks—>Console Graphics

Many people are bored with the console. “It's just text! Console games are stupid!”, they often announce. However, many games are available for Linux which take advantage of console graphics. They use the entire screen, don't require X and don't have silly borders and buttons all around them. Console graphics are fun, fast, and much easier than graphics in X.

The established Linux console graphics library is svgalib, with its sidekick vgagl. Svgalib is a low-level graphics library, and vgagl is a *fast*, frame-buffer-level graphics library for use with svgalib, containing many drawing, text, bitmap, screen buffering, palette handling and 3-D functions. Using these two libraries in conjuction makes programming graphics for Linux exceedingly easy, and both are included in practically all Linux distributions. Although svgalib doesn't work on some cards, needs root privileges to run, and may require an *immediate* reboot or even crash the machine if things go wrong, it usually works and it lets us do much more than X.

Here's a small example of how to get started. Next month, we'll move on to things which look impressive but are just as easy. The full details of svgalib and vgagl can be found with **man svgalib** and **man vgagl**. In case you're interested in a particular function, its man page should be available too; for example, **man vga_waitretrace**. If svgalib doesn't work, install the newest version which supports the new graphics cards. This example opens a graphics screen of 320x200 in 256 colors, draws some shapes, and waits for a key press before exiting. I recommend compiling with this command:**gcc -Wall -O2 shapes.c -lvgagl -lvga -o shapes**

```
#include <vga.h>
#include <vgagl.h>
#define VGAMODE G320x200x256
int main(void)
{
  GraphicsContext *screen;
  char key;
  vga_init();
  vga_setmode(VGAMODE);
  gl_setcontextvga(VGAMODE);
  screen = gl_allocatecontext();
  gl_getcontext(screen);
  gl_setfont(8, 8, gl_font8x8);
  gl_setwritemode(FONT_COMPRESSED);
  gl_clearscreen(0);
  gl_write(16, 68,
    "Console graphics are so cool!");
```

```
    gl_circle(160, 100, 60, 2);
    gl_fillbox(140, 80, 40, 40, 3);
    gl_line(0, 0, 319, 100, 4);
    gl_hline(0, 100, 319, 5);
    gl_setpixel(160, 86, 6);
    for (key=0; key==0; key=vga_getkey())
      vga_waitretrace();
    vga_setmode(TEXT);
    return 0;
  }
```

—Jason Kroll

VENDOR NEWS FROM LINUXWORLD

LinuxWorld was bigger then ever with all the vendors making major announcements. Here are a few of them:

Corel (http://www.corel.com/) presented a preview of its new Linux distribution, called Corel Linux, which will be available in beta in September. Installation is automatic, completing and then asking for special configuration options, such as networking and Ethernet. It includes a GUI for LILO and partitioning of the disk, two to eight virtual desktops, a file manager, new applications built on top of Debian and KDE and easy upgrade facilities.

KeyLabs (http://www.keylabs.com/) came to LinuxWorld to talk about their product testing and certification programs. Testing is focused on hardware compatibility with Linux and is vendor-independent. KeyLabs has been in business since 1996 providing independent, cost-effective testing for the network industry. It is a member of the Canopy Group.

Alpha Processor, Inc. (http://www.alpha-processor.com/) launched its strategic partner program to bring high-performance Alpha applications to enterprise customers worldwide. API provides chip sets and motherboards to manufacturers and is focused on mid-range servers and high-end office workstations. Companies already involved in this program to expand the number of Alpha applications include Cygnus, MySQL, Covalent, Atipa, the LinuxStore and several of the major distributions. Pricing is fast becoming comparable to Intel.

theLinuxStore (http://www.thelinuxstore.com/) launched its PIA (personal Internet appliance) and Element-L Linux-based product line. The PIA provides Internet access, e-mail and word processing, all for $200 US without a monitor. theLinuxStore also offers Alpha solutions from API. It is a subsidiary of EBIZ Enterprises.

Knox Software (http://www.arkeia.com/) showed off its Arkeia backup product on a huge flat screen—the gauges were awesome. Arkeia provides job management, e-mail and a new command-line interface. It is aimed at mid-

range market (ISPs, government, et al.) and provides parallel network backup, multi-tape/multi-node restoration, on-line index, security and a distributed client/server architecture.

Magic Software (http://www.magic-sw.com/) came to the show with two South African penguins last seen in the second *Batman* movie. Magic announced it has ported its e-commerce solution eMerchant to Linux. Previous ports had been done of their development tools that provide a multi-platform database environment to speed development of business solutions. The development kit for Linux is freely available for the single user.

ParaSoft talked about the new versions of CodeWizard and Insure++ that will be coming out by the end of September. RuleWizard, an extension for CodeWizard 3.0, will give developers the option of creating their own rules and will also be out around that same time. A beta version of their new Java testing tool, jtest, will be available in October. ParaSoft products are multi-platform, working on Linux and other UNIX systems as well as Windows.

EARTH-SHAKING HARBINGERS

The Motorola Computer Group is announcing a unified Linux strategy that provides our OEM customers with a broad selection of Linux-based platforms, open-source software, service and support, training and integration services. In support of this broad initiative, we're collaborating with two leaders in the Linux community: Lineo and Caldera Systems. —Noel Lesniak, Business Manager of Linux Telecom Platforms for MCG

The Motorola Computer Group, of which we are a part, has a large emphasis not only in telecom but in other embedded devices. They are a large system-board vendor, both Poser PC-based and X86-based, both of which we'll be targeting with our embedded Linux solution, Embedix. —Brian Sparks, CEO of Lineo

(For complete statements by Mr. Lesniak and Mr. Sparks, as well as Ransom Love, CEO of Caldera Systems, see http://www.linuxjournal.com/articles/misc/005.html).

Pictured (from left to right) are Robbie Honerkamp, Steve Lewis, Jon “maddog” Hall, Greg Hankins, Antoni Dabed, and Reg Charney. Will World Domination be bearded instead of televised? These fellows seem to think so!

LOOSE TALK

When we announced at Lotusphere that we'd be porting Domino to Linux, we got a standing ovation. From ten thousand people. —Don Harbison, Marketing Manager, Notes/Domino Product Marketing, Lotus Development Corporation

We did an internal survey of six hundred people to determine populations at levels of Linux knowledge. At the bottom level you had to know how to spell Linux. All 600 could do that. At the top level, you had to be able to hack kernel code. To our amazement, we had 120 in that group. —Felicity McGourty, Director, Problem Management, Tivoli Systems

Inprise (http://www.inprise.com/), still better known to most developers as Borland, is jumping into the Linux space with both feet. Their own Linux developer survey (which drew respondents from *Slashdot* and *Linux Today*) showed a high degree (72.3%) of interest in Rapid Application Development (RAD) and Integrated Development Environments (IDEs), which have long been a Borland/Inprise specialty. In fact, the top answer to “Which language are you primarily interested in developing in on Linux?” was Inprise's own Delphi (43.9%). The first Inprise product for Linux is VisiBroker for Linux, a new version of the company's popular Object Request Broker (ORB).

Cosource.com (http://www.cosource.com/), the new cooperative market for open-source development, launched a live beta just before LinuxWorld Expo. During the show, the number of proposals to develop open-source projects increased to 10, and financial commitments to the same projects jumped from $50 to $1,640. Driven by the rising interest in open-source development,

Cosource.com expects these numbers to multiply over the next few weeks and months, leading up to the official launch of the service.

SUBSCRIPTIONS

Due to numerous complaints about subscriptions and other unresolved problems with our subscription house, we have brought subscription fulfillment back in-house. We value our subscribers and want them to receive the best service possible.

All information from the outside fulfillment company is now in our own database, with a reworked computer database system specially designed for the task. Visit the *LJ* web page at www.linuxjournal.com/ and click on "subscribe". Our secure form is at www.linuxjournal.com/xstatic/subs/customer_service.html. Using the 8-digit subscriber number on your mailing label, you can inquire about a current subscription, as well as order back issues, *LJ* archive CD-ROMs, renewals and new subscriptions, and pay an invoice. The information displayed will be current as of the time you made the inquiry. Your questions, problems and concerns will now be handled much more quickly, by a knowledgeable staff, using Linux, of course.

Another benefit to subscribers is free access to all back issues on-line at interactive.linuxjournal.com.

Thank you for your patience during this period of transition.

STOP THE PRESSES

Now that Red Hat has gone public, other IPOs (initial public offerings) are on the horizon. We are sure to see one from VA Linux Systems and Andover.net. In addition, don't be surprised to see IPOs from other Linux players over the next year.

As most of us are computer gurus, not stock market gurus, I thought it appropriate to find out how an OpenIPO works. I asked Michael Ackrell of WR Hambrecht + Co to explain it. Here are his responses to my questions.

Q: What is OpenIPO?

A: WR Hambrecht + Co, the new investment bank founded by industry veteran Bill Hambrecht, uses a Dutch Auction method, dubbed OpenIPO, to price and allocate shares in an IPO. Under the auction, orders are received for shares at various price levels. At the end of the auction, orders are accepted starting with the highest bid price and continuing at the lower prices until the number of shares being offered has been sold. Each investor pays the lowest price

accepted, or the clearing price. In addition, investors bidding above the clearing price will receive full allocation at that price. Investors bidding at the clearing price will receive pro-rata allocation. Investors who bid below the clearing price will not be allocated shares. For example, a company files a 2 million-share IPO with a filing range of $12-15. Orders are received as follows: 1 million shares at $18; 500,000 shares at $16; 750,000 shares at $15; 500,000 shares at $14; and 500,000 shares at $13. For this offering, the clearing price is $15. Investors who bid above $15 receive full allocation; investors who bid at $15 receive pro-rata allocation; and investors who bid below $15 receive no allocation.

Q: What are the advantages of an OpenIPO over the traditional approach?

A: OpenIPO, as the name implies, is open to any investor, including large institutions, the over 1,400 small- to medium-sized institutions, and the significant number of retail investors. Institutional investors like the system because they are able to receive the full amount of shares they want, if they bid appropriately. Retail investors like the system because it provides them access to IPO shares, and their orders count the same as those from institutions. Issuers benefit from the system because their stock is available to a much wider group of potential investors. Furthermore, the auction attempts to establish a more efficient pricing environment, one that captures the true demand for a stock in the IPO price rather than in the aftermarket.

Ultimately, through OpenIPO, the market, not the underwriter, prices the IPO. While the system will not alleviate all aftermarket price fluctuations, it should put more money into the hands of the issuer.

For the Open Source community, OpenIPO is a way for developers to actually receive shares in an IPO. Under the traditional approach, they would not be able to get any shares. OpenIPO provides better pricing for the issuer and better allocation of stock. Red Hat left a lot of money on the table, as the price rose significantly on its first day of trading and it was unable to get stock into the hands of its developers.

Vendor News

Cygnus Solutions announced plans to release the source code to Cygnus Insight, a graphical user interface (GUI) for the industry-standard GNU debugger, GDB. Known in programming circles as GDBtk, the Cygnus Insight GUI provides the technology for effective and efficient debug sessions by improving a software developer's ability to visualize, manage and examine the status of a program as it is running. The source code for Cygnus Insight debugger will be available from Cygnus at http://sourceware.cygnus.com/gdb/.

Kasten Chase, a provider of host access connectivity solutions announced that its VersaPath web-to-host product (http://www.versapath.com/) will support the Linux operating system and Java client technology, making this complete web-to-host solution more flexible. VersaPath fully integrates desktop and browser clients in a single solution.

Metrowerks Inc. (http://www.metrowerks.com/), a supplier of software development tools for telecom, desktop, embedded systems and consumer electronics, and SuSE GmbH (http://www.suse.com/), a provider of Linux software and consulting services, announced a partnership to provide CodeWarrior software development tools for the SuSE Linux operating system.

The Debian Project announced a hardware vendor commitment from **Linux Laptops Ltd.** Linux Laptops is the only hardware vendor devoted exclusively to delivering portable computers with Linux software installed and ready to use. Laptops with Debian GNU/Linux pre-installed can be ordered via the company's web site at http://linuxlaptops.com/.

Stormix Technologies in Canada announced the alpha version of a new Linux distribution called Storm Linux. Based on the Debian GNU/Linux distribution, Storm Linux is designed to be easy to use and simple to install. Its target market is both the server and the desktop market. All development for Storm Linux will be open source. The final release of Storm Linux is expected in the fourth quarter of 1999.

Red Hat, Inc. (http://www.redhat.com/), a developer and provider of Linux-based operating system solutions, announced that Global Knowledge, an independent IT training company, will provide hands-on, real-world training and certification nationwide for Red Hat Linux, including the Red Hat Certified Engineer program.

Hummingbird Communications Ltd. (http://www.hummingbird.com/), an enterprise software company, announced it has joined Red Hat's Independent Software Vendor Program and will participate in joint marketing activities. The new relationship will give Linux users access to all of Hummingbird's fat-client connectivity products including Exceed, HostExplorer, NFS Maestro Server, NFS Maestro Client, NFS Maestro Gateway and NFS Maestro Solo.

Stalker Software, Inc. (http://www.stalker.com/) announced the LinuxPPC version of their high-end CommuniGate Pro messaging system. CommuniGate Pro is a Unified Messaging Server which supports most major operating systems. On all platforms, CommuniGate Pro presents the same interface and uses the same file formats, allowing any organization to switch server platforms in less than an hour.

SourceGear Corporation (http://www.sourcegear.com/) announced today that it has acquired Cyclic Software and is looking forward to the opportunity to be involved in the support and development of CVS. SourceGear is a new identity for an existing company and is the founder and sponsor of the AbiWord project. They are active participants in the free software world and intend to serve the community as active maintainers of CVS and leaders in the ongoing development of this tool.

Oracle announced it was developing a computer (Intel-based) with no hard drive that will sell for $150 without a monitor, $250 with a monitor. It will come with the Linux operating system and Netscape Navigator installed. A CD-ROM drive will be used for booting the system and upgrading software. Dates for its manufacture and release have not been set.

Archive Index Issue Table of Contents

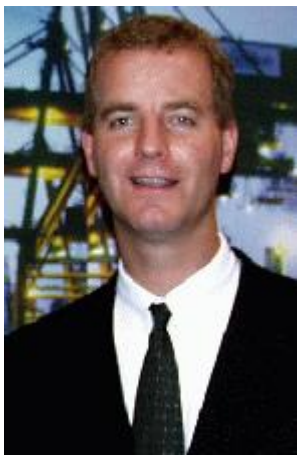
Advanced search

Interview: Lyle Ball, Lineo

Marjorie Richardson

Issue #66, October 1999

I talked to Lyle Ball of Lineo to find out which rumors were true.



On July 20, Caldera Thin Clients announced it was changing its name to Lineo and would be offering an embedded system product called Embedix, based on Caldera's OpenLinux. Then the rumors began to fly—who were they partnering with? The name heard most often was Motorola. On August 4, under an NDA until the August 9 announcement, I talked to Lyle Ball of Lineo to find out which rumors were true. Here's our conversation.

Margie: Sounds like exciting things are going on at Lineo. Tell us about it.

Lyle: On Monday, we will be publically announcing a strategic relationship between two Linux companies. Caldera Systems will be providing Motorola with OpenLinux for their high-availability network solutions, and for now, Lineo will be the only company working with Motorola in this way. We will be providing them with Embedix, which is Lineo's Linux platform for the embedded market.

Embedix is exclusively based on OpenLinux, and that exclusivity works both ways. Caldera Systems will not be partnering with anyone else in the embedded space and will not be providing the embedded Linux business—that will all be

taken care of through Embedix and Lineo. We will not be using anyone else's solution. We won't, for instance, offer a Red Hat Linux solution as an alternative. So, from a technology standpoint, we will be taking Caldera Systems' OpenLinux platform into the Embedix arena. From a business standpoint, it is Lineo's Embedix that will be branded in that arena. Is that confusing or clear?

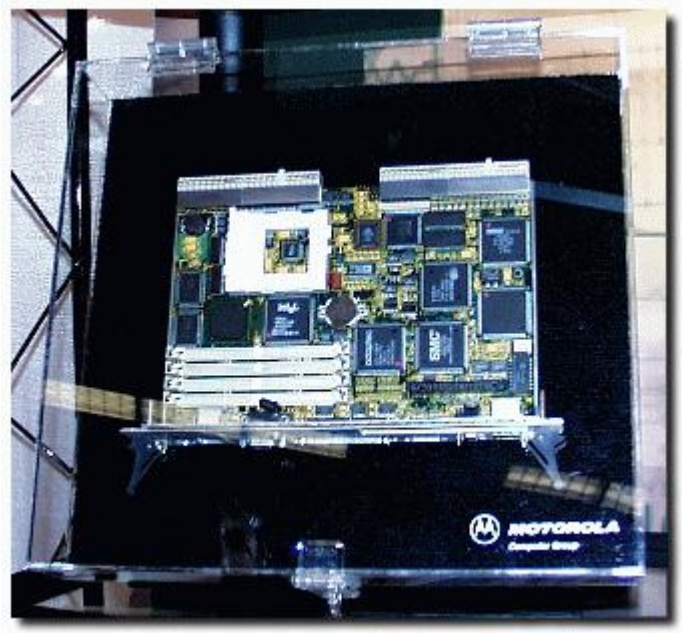
Margie: Very clear, you said it well.

Lyle: The big news here is that the world's largest provider of embedded technologies, Motorola, has selected Lineo and Caldera Systems as the software partners to launch Motorola's aggressive Linux initiative. Motorola will be providing Linux-based solutions from top to bottom—from their larger high-availability solutions all the way down to their smallest chips. Lineo will be providing the smaller space pieces, i.e., the embedded software pieces, under the brand name Embedix, and Caldera Systems will be directly providing the OpenLinux product for the high-end box, high-availability Linux solution.

Margie: This really sounds like hot stuff!

Lyle: It is big for us and we think it is really big for Linux. Companies have invested in other companies before—Red Hat has provided great exposure for Linux in investments. I personally believe this very high-level delivery of products presents a different perspective. For a world-leading embedded provider of hardware to make the investment and the commitment to target Linux—not just announcing a strategy, but actually doing it—is a big story. We will be doing just this as partners—actually presenting the products that will be delivered with their solutions. I'm very excited about Monday (August 9) and all the Linux news, such as Red Hat going public and our announcement. I'm sure a ton of other announcements will also be made at Linux World; it's going to be a very good Linux week. We are excited to contribute to that with this announcement.

We have been fighting hard for years. I've been in this for five years now at Caldera and then on to Lineo, and I'm just very excited this is coming out—moving forward.



Margie: How long have y'all been working on this embedded thing—not for five years, surely!

Lyle: Not for five years, but Lineo has been working on it for some time. The product is in progress, it's not a philosophy, it's not a white paper, it's product. If you want to understand our product, a press release I wrote is at www.lineo.com and talks about renaming Caldera Thin Clients to Lineo and the Embedix product. Embedix has over 130 components, and many of them are complete. OEMs (original equipment manufacturers) will build solutions based on any combination of those components. We already have OEMs working with these—Motorola is the first that is ready to announce. In the embedded OEM space, they don't announce their intention until they basically have the product done, because they don't want their competitors to know what they are doing.

Margie: I have seen your announcement, and my question is: which OEMs are already signed up?

Lyle: We can't say. It's not like in the software-only arena where Microsoft will vaporware their product a year and a half before they build it. Because of their dominance, they can do that. In the hardware arena where things can be copied so quickly, software components can be added into existing hardware. They don't want to talk about it until they are close to shipping it. Motorola is our first OEM to announce, but it is definitely not our last!

Margie: It's definitely a big one, too. What kinds of tools are included in Embedix?

Lyle: In general terms, we have taken Linux and optimized it for three areas required by OEMs: memory, storage and performance. We have optimized Linux technologies, in our case based exclusively on OpenLinux, to meet the rigorous demands in the restricted embedded space, where memory, storage and performance are all very critical. For instance, in the software desktop and PC arena, there is basically unlimited memory and speed and storage at this point. You can add on another terabyte or gigabyte, whatever you want—it costs, but you can do it. In the embedded space, the idea is how small you can get it, not how big. It's not how many more Linux packages I can add to make my product bigger and better than my competitor. It's how much tighter and smaller I can make my product and still have it do more than my competitor. It's the exact opposite philosophy of where Linux has been going in the last few years.

Margie: Did you have any problems with getting it stripped down to size?

Lyle: Of course, there were problems! Engineering is not an easy task! But we feel we are well ahead of any known competitors. First of all, we don't know of anyone of significance who has announced their actual product plans. A couple of small companies have announced they are going to try embedded Linux initiatives in software. None of them have a large partner like Motorola, to our knowledge. Again, we might be surprised on the 9th!

We do believe we are the first to market with embedded solutions, and we will aggressively define the embedded Linux space. We won't be the only one doing that, but we believe we will be integral to defining that space. And we believe we have a great chance at maintaining market leadership in that space, because we have been doing embedded for a few years and Linux for five. It's a unique evolution for Caldera Thin Clients, where we have a Linux heritage and a Linux sister company. We also have very deep experience with embedding DR DOS, and we've been making millions from that. So we are in a unique position: we are not a startup and we have funding. Our DOS product paid for all our R&D on embedded Linux. We have the processes in place, we have the employees, and we are simply evolving our focus from an embedded DOS-only company to an embedded Linux company.

Margie: Have y'all been doing this for a while?

Lyle: Oh, yes, it's been in the plans—just not public. You don't want your competitors to know everything you are doing.

Margie: How about the name change? How did you choose Lineo?

Lyle: We wanted a name that was more attached to the Linux environment, because we see a longer-term demand for Linux from the market. We are not killing our DOS product immediately; that is, the market is not killing our DOS product. There is still a high demand for embedded DOS, and we will continue to sell and market it. However, there has been an increasing demand for embedded Linux. So we are shifting our focus and renaming the company to match our longer-term revenue stream, which will be Linux-based.

Let me make it clear that we are not abandoning DOS, we are just shifting as the market has requested us to do. No one said, “Kill DOS now!” They said, “Hey, we need DOS, but, long term we would really like to have these Linux solutions as well.” It will be up to each individual OEM as to when they will switch over. Motorola is doing it now, and others will do it later. We will keep selling both technologies during the transition.

Margie: This may be a rather obvious question: when Cygnus decided to get into the embedded business, they did their own operating system and said that Linux wasn't going to work for them...

Lyle: Too bad for them!

Margie: What choices did y'all go through; did you decide to use Linux just because you had OpenLinux right there, or did you have other reasons?

Lyle: Well, one of the reasons was Caldera Systems has been very effective in building a solid brand around OpenLinux. OpenLinux is known as “Linux for Business”, and it's got this corporate and trusting feel to it. There may be weaknesses the product has, as all products have strengths and weaknesses, but its strength is that it has a solid name in the business community. Caldera has spent—I don't know if they have ever disclosed how much—an insane amount of money in advertising and other promotion to build Linux itself and OpenLinux as acceptable in a business setting. Therefore, when we spoke to our OEM companies—not just in the U.S., but around Europe and Asia—they were interested in our DOS solution and they would also say, “Boy, we would like an OpenLinux-based solution. We would like to switch to Linux, and we want to switch to a Linux that has a name and is comfortable and established. That way, we aren't perceived as bringing something new to the table—it's an evolution rather than a startup.” So the philosophy is that Caldera Systems has done a phenomenal job of building market awareness and a solid brand. Embedix is the business brand that Lineo will build, because we need our own brand. But it will always be accompanied with the tag “exclusively based on OpenLinux and licensed from Caldera Systems”. Embedix isn't licensed from Caldera Systems; Lineo owns its enhancements. Just as Red Hat and Caldera Systems always give credit to Linus for the Linux they have built their systems

on, we will always give credit to OpenLinux for the base. We will call our additions—we actually call them “additions” even though we are making OpenLinux smaller in size; it's definitely advancements because it takes a lot of engineering to do so—our enhancements, “Embedix”.

Margie: Thanks for your time.

Archive Index Issue Table of Contents

Advanced search

GNUPro Toolkit for Linux v1.0

Daniel Lazenby

Issue #66, October 1999

GNUPro Toolkit is a set of tested and certified, open-source, GNU standard C, C++ and assembly language development tools.

- Manufacturer: Cygnus Solutions
- E-mail: info@cygnus.com
- URL: http://www.cygnus.com/
- Price: $79.00 US
- Reviewer: Daniel Lazenby

GNUPro Toolkit for Linux is designed for developing both commercial and noncommercial Linux applications on native Linux platforms. It is a set of tested and certified, open-source, GNU standard C, C++ and assembly language development tools. The reviewed release of the product (v1.0) is a Linux-only product. This packaging of the Toolkit is specifically targeted toward the Linux desktop developer.

GNUPro Toolkit Suites

The compiler and development tools consist of GNU's C compiler (ANSI conforming), C++ compiler (ANSI tracking), assembler, C preprocessor, linker,

debugger and the GNUPro visual debugger. The GNUPro libraries include standard ANSI C runtime, math subroutine libraries, C++ class and C++ iostreams. All of the compiler and development tools, except **gdbtk**, are command-line tools. (**gdbtk** brings up the GNUPro visual debugger). The other command-line tools function like standard GNU tools.

A set of binary and general utilities is also included. Tools such as objcopy, objdump, ranlib, strip, ar and nm are examples of the included binary utilities. The general utilities provide a set of tools for comparing and merging files. Tools such as cmp, diff, diff3, sdiff and patch are included in the general utilities.

Visual Debugger

For me, GNUPro's visual debugger makes debugging easier. I like being able to see graphically what is happening. Figure 1 contains a KDE environment screen capture of the debugger's Source Window.

Figure 1. Source Window for the Debugger

Typical menu, tool, display window and status bars are provided. The tool bar provides run, stop, function navigation and assembler navigation buttons. Then there are the register, memory, stack, watch expressions, local variables, break points and console dialog box buttons. These buttons are followed by the stack navigation buttons. Each of the function buttons implements or executes a GNU gdb command-line interface command. The dialog buttons open windows that display related information. The Register Window button dynamically displays registers and their content. Memory is dynamically displayed in the Memory Window. The current state of the call stack is displayed in the Stack Window. The Source Window will update its display to reflect any selections made in the Stack Window. An “expressions to watch” window is displayed by the Watch button. This window will be blank unless expressions, registers or pointers have been identified for watching. Current values of local variables are displayed in the Local Variables Window. A list of all defined breakpoints is shown in the Breakpoint Window. Another way of managing breakpoints that may not be readily visible/accessible in the Source Window is provided by the Breakpoint Window. A Console Window is displayed by the console button. This window provides a command-line interface to GNUPro Debugger.

A mouse click or static cursor within the Source Window produces several pieces of information. The current value of a variable is displayed when the mouse cursor is held over a variable. Holding the cursor over a variable and right-clicking the mouse produces a pop-up menu. This menu offers two choices: add the item to the watch list, or dump the memory. Selecting either of these produces a dialog box displaying the selected information. Executable

lines are shown with a minus sign on the left side of the window. Left-clicking the mouse in this region will either add or remove a break point. Right-clicking the mouse offers a pop-up menu with a couple of break point management options.

Below the status line are three drop-down list boxes. All of the source and header files associated with the executable are presented in the left-most box. The middle box displays all functions in the source or header currently displayed. Any of the items in these two list boxes may be selected for display. Four items in the right list box control what is being displayed. The display options are source only, assembly only, source and assembly mixed, and a split screen with source on top and assembly at the bottom.

GNUPro Toolkit Installation

Version 1.0 for Linux of GNUPro has been tested and is officially supported on Red Hat version 4.2 and 5.x. GNUPro also works with Red Hat version 6 and SuSE version 6.0/6.1. Plans for the next release of GNUPro for Linux include testing and support on distributions other than Red Hat and SuSE. The install files provided on the review package distribution CD-ROM media were for Red Hat versions 4.2 and 5.1.

Installation of GNUPro on Caldera's v1.3 required a little experimentation, some research and thinking. Cygnus FAQ pages indicated GNUPro should install on Caldera, Debian and other Linux distributions. With this positive reinforcement, I began installing the product on a 300MHz Pentium with 64MB RAM and Caldera's OpenLinux version 1.3 with KDE.

The Red Hat version 5.2 install file produced a list of missing dependencies. It turned out the Red Hat v4.2 install file had just one missing dependency. The version 4.2 install file was looking for libncurses.so.3.0. OpenLinux had a later release than the install program expected. The install program would not accept either a soft or a hard link to libncurses. The Cygnus bug list web page suggested a way around this specific problem. I got around ncurses hurdle by installing GNUPro using the **nodeps** option. After the install had completed, I made a soft link between libncurses.so.3.0 and libncurses.so.4.1. Once the link had been established, the installation verification tests executed successfully .

Documentation and Support

In general, Cygnus support was prompt, responsive to my questions and to the point. In response to one question, they provided some additional information in case I might be considering upgrading to glibc 2.1.

According to the GNUPro Getting Started Manual and FAQ, HTML copies of the six GNUPro documents (Compiler Tools, Debugger, Libraries, Utilities, Advance Topics and Tools for Embedded Systems) were included on the distribution CD-ROM. I was unable to locate any HTML documentation on the CD-ROM. After roaming about the GNUPro installation directories, I located HTML documentation files in my install directory /usr/cygnus/redhat-980810/doc. Cygnus support indicated that the User Guide's reference to documentation is being corrected.

The on-line documentation is in HTML format. It requires a web browser or some other application able to read HTML files. On my system, I chose to use **kfm** to access and display the GNUPro HTML documentation.

GNUPro Toolkit for Linux is a single-release snapshot of the ever-evolving GNUPro product. Support for the Linux product includes 30 days of installation support and a web page. The support web page contains PDF and HTML documentation, FAQs, patches and a known bug list. E-mail notification whenever the patch or bug list pages change may be requested.

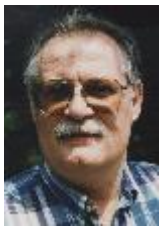
The 30-day support is for installation problems. After a successful product install, one should not expect a personal reply to non-installation support questions. At the time of this writing, the primary after-installation support vehicles are the Cygnus web site, newsgroups and mailing lists. No information on upgrade policies for GNUPro for Linux was available.

A quick comment on GNUPro legal notices. GNUPro Toolkit consists of software from several different sources. The terms and conditions for copying, modifying and redistributing software components derived from GNUPro vary. The User's Guide devotes a short chapter to the various legal notices.

GNUPro Toolkit, the Full-Featured Product

While this review was specifically about GNUPro Toolkit for Linux, I would be remiss if I did not briefly mention Cygnus' GNUPro Toolkit product. The more robust, and more expensive, GNUPro Toolkit is capable of supporting more than 125 hardware and software environments. A wide range of host systems may be used to develop 32- and 64-bit applications for this array of target hosts and operating systems. Some of the many major hardware and software players represented include HP, SGI, Sun SPARC series, Hitachi, MIPS, NEC, Panasonic, Toshiba, Motorola, IBM RS6000 and PowerPC and Intel's X86 series, to name a few. The Toolkit contains embedded cross-platform compiling capabilities. No cross-platform capabilities were included with the reviewed Toolkit for Linux version. Cygnus has identified Linux as one of its target host environments, so cross compiling should be on the horizon.

Cygnus offers a GNUPro subscription service for the high-end GNUPro Toolkit product. This service provides two releases of GNUPro each year, and access to Cygnus Developer Support. Additional information on pricing and Developer Support services may be obtained by contacting Cygnus.



Daniel Lazenby first encountered UNIX in 1983 and discovered Linux in 1994. He can be reached at d.lazenby@worldnet.att.net.

Archive Index Issue Table of Contents

Advanced search

Fast Ethernet Network Starter Kit (FENSK04)

John Kacur

Issue #66, October 1999

The kit includes two EtherFast 10/100 PCI LAN cards, two 15-foot category 5 UTP cables, an AC power adapter for your hub and a 4-port 100Mbps fast Ethernet hub.

- Manufacturer: Linksys
- URL: http://www.linksys.com/
- Price: $109.95 US
- Reviewer: John Kacur

I was shopping in one of those computer megastores, looking for a network kit for a project of mine, when I spotted the Fast Ethernet Network Starter Kit from Linksys. What caught my eye was that Linux was mentioned right on the box. As I didn't have the list of Ethernet cards known to work under Linux with me, I decided to take the words on the box at face value and give it a try.

The kit includes two EtherFast 10/100 PCI LAN cards, two 15-foot category 5 UTP cables, an AC power adapter for your hub and a 4-port 100Mbps fast Ethernet hub. The kit also comes with an instruction booklet and drivers for Windows 95, Windows NT 3.51, Windows NT 4.0 and NetWare. This is all good, I thought, dreaming of the fun experiments I would have with SAMBA; however, I

bought the kit because it promised compatibility with Linux (albeit listed in the “Others” category on the box).

I didn't see Linux mentioned in the instruction booklet, so I took a look at their web site (http://www.linksys.com/). In the FAQ, I found www.linksys.com/support/solution/nos/linux.htm (and freebsd.htm too, for our free UNIX brethren).

The amount of Linux information on their web site has grown since I first looked at it, but this is probably not your best resource. They mention, for example, that if you're installing Red Hat 5.2, you should choose the **Tulip** driver from the list of drivers on your screen. While this isn't bad advice, it might give the false impression that you need to re-install Linux to get your Ethernet card to work. They also mention the driver has been tested under SuSE, Caldera, Slackware and Debian, when use of the driver is, of course, absolutely distribution independent.

The tulip.c driver is supplied on one of the floppies in the kit, and there is a link to the most up-to-date version. This version is found on the CESDIS (Center of Excellence in Space Data and Information Sciences) web site and, like so many Linux Ethernet drivers, is written by Donald Becker. In my opinion, your best resource for the driver is cesdis.gsfc.nasa.gov/linux/drivers/tulip.html. Also, Greg Siekas deserves a mention for his very clear instructions found at www.bmen.tulane.edu/~siekas/tulip.html. This page also has information on different options for people compiling the tulip driver for different cards.

Listing 1.

In order to get instructions on compiling the driver, type:

```
tail tulip.c
```

The output is shown in Listing 1. Notice there is a slightly different syntax for SMP (more than one processor). Most people will want to use something like:

```
gcc -DMODVERSIONS -DMODULES -D__KERNEL__\
-I/usr/src/linux/net/inet -Wall\
-Wstrict-prototypes -O6 -c tulip.c
```

Next, append this information to the /etc/conf.modules file:

```
alias eth0 tulip
options tulip options=11 debug=0
```

Setting **options** to 11 sets the media type to MII autoselect, and setting **debug** to 0 suppresses the debug messages. Set **debug=6** if you want to obtain the very wordy debug messages.

Finally, copy the object file to the latest kernel's modules:

```
cp tulip.o /lib/modules/2.X.XX
```

and update the kernel dependencies:

```
depmod -a
```

If you are using the driver in a monolithic kernel, then copy tulip.c to the /usr/src/linux/drivers/net directory and recompile the kernel.

The Linksys Starter Kit performed beautifully under Linux and I haven't had any problems using it with some older PCs, but be aware that you can't set the IRQs with a switch. Your best bet is to use a fairly modern PC with an up-to-date BIOS that can automatically configure your card's parameters.

My only complaint with this kit is a minor one—I find the 4-port hub to be a bit limiting. You can uplink the hub with others, but doing so makes port number 1 unavailable for a PC. However, Linksys now has a new Network Starter Kit which includes a 5-port hub (FENSK05).

I imagine Linksys is targeting home users wishing to do some networked gaming and to connect more than one computer to the Internet at a time. Toward this end, Linksys has a special offer which includes a two-user version of Virtual Motion's Internet LanBridge with the kit. I would like to point out to new Linux users that Linux can be used for free to connect many computers, running any operating system, to one dial-up account.

In short, I have no problem recommending the Linksys starter kit to Linux users. You should be prepared to do a little more work than expected by people from the plug-and-play world. It is nice to see a growing number of companies which are starting to support Linux users.

John Kacur (jkacur@vaxxine.com) is using the starter kit for his thesis, “Mini-Beowulf”, in which he demonstrates the principles of parallel computing on a small four-machine cluster. His project page is at http://www.vaxxine.com/johnk/beowulf/.

Archive Index Issue Table of Contents

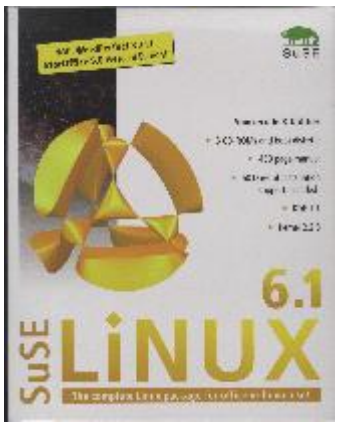
Advanced search

SuSE Linux 6.1

Jason Kroll

Issue #66, October 1999

This is a serious, high-performance distribution which is more complete than practically any single distribution, and at the same time is clean and fast due to effective configuration and intelligent design.



- Manufacturer: SuSE GmbH
- E-mail: info@suse.com
- URL: www.suse.com
- Price: $49.95 US
- Reviewer: Jason Kroll

Hailing from Nuremberg, Germany, SuSE Linux 6.1 is SuSE's latest release (SuSE Linux 6.2 will be available by the time you read this) featuring the new 2.2 kernel, custom SuSE software, commercial packages (Netscape, StarOffice with personal license, Applixware Office, Corel WordPerfect 8.0 and many others), five CD-ROMs' (and a disk's) worth of software, a 440-page manual and a green chameleon sticker. This is a serious, high-performance distribution which is more complete than practically any single distribution, and at the same time is clean and fast due to effective configuration and intelligent design. Much attention has been paid to the details in this release (as in past releases), and the production gains a lot from SuSE custom software and configuration.

Although SuSE is the top Linux distribution in Europe, and the 1998 *Linux Journal* Readers' Choice for “Best Distribution”, it has not been particularly successful in the U.S. The reason for this is not clear (perhaps someone in marketing knows), but if word gets out about this distribution, the situation is likely to change.

YaST

Yet another Setup Tool is SuSE's software for installation, de-installation and maintenance. Once a system is installed, YaST stays around to help with many configuration tasks, and updates system files accordingly. (For example, if you want to change networking configuration, YaST asks what changes you want and automatically updates all files.) YaST is also the tool that installs the whole system in the first place.

Installation of SuSE is quite easy and extremely flexible, although not entirely automated—SuSE begins with a default (base) package and allows you to modify package selection to suit your needs. You can also load a few pre-configured selections (with the usual server, workstation, and complete installation options), though one ought to customize even these configurations. It is easiest just to have three or four pre-configured installations (as with Caldera), but SuSE probably has a number of reasons for taking the custom installation approach. For one thing, SuSE is more complete than most other distributions (five CD-ROMs is a lot of software, and the new 6.2 apparently has six) and customization makes more sense when dealing with so much software. Also, SuSE does not aim to be “Linux for Idiots” and would lose much of its flexibility were it dumbed down for people who can't decide what software they want.

One benefit of SuSE's devotion to custom installation is that SuSE has developed a system which incorporates many packages into the menuing system of KDE. In addition, YaST knows what dependencies the packages (numbering about 1000 for 6.1 and 1300 for the brand-new 6.2) have, and can *automatically* install these dependencies. YaST keeps track of redundancies to warn a user against installing software packages that are too similar or would be unused. Also, YaST has good descriptions of the software packages, so you always know what you're going to get (well, except for the occasional !! HIER FEHLT DAS LABEL !!). In these areas, SuSE is unique. For example, Red Hat offers far fewer packages and mostly does not incorporate selected programs into menuing systems. Caldera also offers fewer packages, and while it effectively incorporates programs into KDE, it offers four installation packages and no custom option. However, Caldera's installation program, Lizard, is probably the easiest.

Installation can take place in over a dozen languages (English is second on this list). Before installation begins, you have to load any special drivers you might need. Secretly, I wish SuSE would autoprobe, and in fact there is an automatic detection for necessary modules which actually works, but you have to find it first. Once the modules are taken care of, proper installation can begin.

Ultimately, installation is straightforward and easy to navigate, but, like Red Hat, often requires you to know what hardware you have. I would prefer more probing and perhaps more complete default installations (I had to select so many packages, it took a while), but there were no technical hang-ups, freezes or crashes. Manual configuration of networking was just like most distributions, and the system was booted and on-line fairly quickly.

SaX

SuSE Advanced X Configuration Tool (SaX) is a graphical interface for configuring X (and we all know how much of a pain that can be). You can navigate its menus with the mouse or the keyboard, and all you have to do is choose your mouse, graphics card and monitor from a few lists and an XF86Config file is created. I noticed that the lists were not as long as I have encountered elsewhere, and some older hardware was missing. SaX configures your keyboard and mouse as well and offers many expert configuration options.

Once you have finished with basic configuration, SaX allows you to test the video modes and make real-time adjustments, rather like **xvidtune** but graphical and easier to use. You get to see the standard X test pattern for a little while, and that's nice, but I would still prefer keeping it up all the time to get a better idea of how well the screen is configured. It's very simple to use, and if it doesn't work, you can always try the xf86setup or xf86config. Even after your system configured, you can use SaX to update your configuration in case you would like to use higher resolution, a new video card or a new monitor. This is unlike other installers which configure once during installation and require the use of external software or manual modifications afterwards. Once the XF86Config file is created, it's time to try out X.

The X Window Manager

The default window manager on SuSE Linux 6.1 is KDE, which has, in addition to its standard menus and KDE software, a SuSE menu housing entries for many of the various packages installed by YaST. Also configured in the menuing system are the various Applix programs, though at the moment they insist the license is out of date. (I hope this is corrected in 6.2.) SuSE's custom menus have more to offer than the custom menus of most other distributions, even though some menu options don't work and many installed software packages

are not incorporated into the menuing system. Still, it is a clever idea to have YaST configuring KDE's menus during installation in such a way as to have optional packages included. If SuSE had more time to devote to going over the packages and making sure they all were properly collected into the menu tree, the system would be really neat. (The next step would be automatic real-time updating of the menu trees to correspond with the software on the system...)

Screen Shot

On the whole, SuSE's window manager works just fine. You might have to know enough during installation to select the right XF86 server, however. SuSE's menuing system looks quite neat, and for the most part, files seem to be in the right places. No menu selections I found led to core dumps or worse, and although many of the packages I installed did not get entered into the menus (and some selections didn't do anything), you can do a lot with KDE as installed on SuSE. If you like GUIs (I, for one, vastly prefer consoles), KDE is very functional.

Technical Concerns

It is difficult to say anything bad about a Linux distribution; after all, they're all based on Linux and GNU software. Nevertheless, concerns sometimes arise over the decisions made about which programs and versions are used, and sometimes things just don't work. Where installation is concerned, auto-installers are nice when they work, but probing often crashes the system. On the other side, many people have no idea what hardware is inside their system and what IRQ and DMA values they ought to enter. Preconfigured package collections are convenient but lose flexibility, whereas manual selection of packages takes forever. Experimental software usually has enhanced functionality over stable software but often fails unexpectedly, as many of you know who simply *must* run the latest kernel/library/compiler, even when parts of it aren't working.

The choices to be made by a distributor can be difficult, and the practice of targeting a specific audience guides distributors in their decisions. Although I've recently been recommending SuSE when asked by newbies which distribution to install, I am aware it's not for computer illiterates. At the same time, it *is* easy to install and use and is rather well-endowed. It appears to be an ideal distribution for the middle 68% of Linux users, often choosing ease-of-use and stability over the cutting edge, while at the same time favoring maximum functionality and flexibility against oversimplification. The installation and configuration utilities (YaST and SaX) are examples of the latter, while the choice between gcc and egcs is an example of the former and latter, respectively.

I tried compiling a number of “problem” packages, sources that have always “bugged out” when I tried to compile them in the past, and every single time, SuSE compiled without error. Obviously, SuSE must have functional libraries in the right locations. (This was a problem with certain distributions years ago.) The kernel has never been one of those problem packages, and recompilation/installation went very smoothly. SuSE has a KDE menu item which starts up the kernel recompilation—another clue that SuSE is geared towards non-neophytes. Upon reboot, everything was in the right place and all the modules worked. This probably has more to do with kernel developers than with SuSE, but someone truly clever could have contrived a way to stop a kernel and its modules from working. In any event, kernel recompilation worked even better on SuSE than on Red Hat (which had some complications involving modules) and Caldera (which isn't targeted at kernel hackers anyway), although Slackware has never presented me with any problems.

SuSE Linux 6.1 has a lot of small things going for it, such as a different color scheme for the **ls** command, syslog messages logged to **alt-F10**, a first console which does not clear the screen on logout/login (so you can still read system messages from startup, etc.) and various other details. Unlike Red Hat, SuSE does not stick its logo all over the desktop (or on every single window, as many of you may remember). On the whole, SuSE has a cool, modern feeling to it—it isn't obnoxious. SuSE has not bent over backwards to make system configuration easy for the command-line impaired, so the system itself is not overly complicated or obtuse, yet it still works. And if it doesn't work, there's always support.

Support

The general failing of most distributors is poor support. However, Linux firms have recently been putting increased effort into providing better support, and third parties such as Linuxcare are raising standards in the industry. Support from SuSE comes not only as “tree-ware”, but as 60 days of e-mail/phone support. If, after 60 days, you can't get your system installed, buy a Mac or try one of the new pre-installed systems. I have not yet heard a complaint about SuSE support, and I expect that SuSE must be competent in this department—SuSE claims 35,000 business customers (including Mercedes-Benz), and businesses have a low tolerance for bad technical assistance.

The best support in my opinion is an excellent manual, and SuSE has done a superb job here. Just like SuSE's custom software (and the distribution itself), the manual is a product of thoughtfulness, attention to detail and cohesive design. The book is 440 pages, but quite dense—nothing is watered down, and even simplifications have footnotes explaining what the reality is. An example of this simplification and explanation approach is that the installation guide has a section for “quick install”, as well as an in-depth section on the whole

installation procedure and its options. The book is also not condescending, and I find that refreshing.

The manual is divided into eight parts: Introduction, Install SuSE Linux, Network Configuration, The X Window System, Linux and Hardware, The Kernel and Its Parameters, SuSE Linux: Update and Specialties, and Security and Hints. The last of these chapters actually contains an introduction to Linux as well as the immortal line, “natural disasters such as lightning strikes, floods and earthquakes can damage your computer”. There is a bit of humor and some math in the book, and a noticeably “green” feel: the computer system in the book is named “earth”, and the manual says of the free books included in PostScript format, “If you don't care about trees, you can print them as well.”

Conclusion

SuSE Linux is a complete Linux distribution aimed at non-neophytes or perhaps neophytes who are good with computers. Although the price might suggest it is worth less than a copy of Windows (or some very expensive Linux distribution), it is quite difficult to do better for the money. Like any Linux distribution, there are menu options which don't work and software that's missing from the menu selections. The default package collections are good enough, but it's worth the time to step through every package and decide whether or not you want it. If you're going to get a commercial Linux distribution, you'll do well to consider SuSE—at the very least, you get several CD-ROMs' worth of software, good news if your CD-ROM drive is faster than your Internet connection. When the new SuSE Linux 6.2 arrives (with the 2.2.10 kernel and a whole bunch of new software), it will definitely be one to check out.

News Brief

Jason Kroll spends his weekdays in the Product Testing Lab at *Linux Journal* where he is very happy to be working in support of the Linux movement. His evenings are supposedly spent finishing his economics studies (but not really, other classes are more fun, you see). When reviewing a distribution, he thinks it's very important to test all of the games and wishes distributors would include more games with their distributions. He can be reached at info@linuxjournal.com.

Archive Index Issue Table of Contents

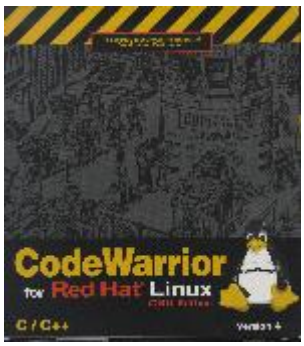
Advanced search

CodeWarrior for Red Hat Linux, GNU Edition, Version 4

Jason Kroll

Issue #66, October 1999

Metrowerks is releasing two versions of CodeWarrior for Red Hat Linux (with SuSE releases to follow): the GNU Edition and the Professional Edition. The GNU Edition is the subject for today.



- Manufacturer: Metrowerks
- E-mail: sales@metrowerks.com
- URL: http://metrowerks.com/
- Price: $99 US
- Reviewer: Jason Kroll

CodeWarrior is an integrated development environment (IDE) which has existed on several platforms for a number of years and has just recently been made available for Linux. Metrowerks is releasing two versions of CodeWarrior for Red Hat Linux (with SuSE releases to follow): the GNU Edition and the Professional Edition. The GNU Edition is the subject for today.

The IDE

An integrated development environment, as in the case of CodeWarrior, is a graphical environment incorporating the main elements of software development. This includes managing the project, dependencies, libraries, editing code, compiling, debugging, linking and all of the usual coding tasks. It

does not include paint programs or other multimedia tools that could be used in software development. Essentially, the idea of the IDE (not to be confused with Integrated Drive Electronics) is to combine all these things under one interface for easy project management. This is a different approach from using Emacs in one virtual console while running **gcc** in another and editing the Makefile in a third. An IDE will presumably take care of the entire project, and one can easily call up whichever parts of the program need editing and have the computer keep track of changes to the Makefile (or what would be the Makefile). This approach has its advantages and disadvantages.

The Advantage

The principal advantage of the IDE is that it is *easy*; the program takes care of everything and lets you concentrate exclusively on your code. On machines without the **make** command, this can be completely wonderful, and even on Linux, many users have wished that someone would port CodeWarrior. Now it is available, and the development environment is basically the same as on other platforms, so migrating from other operating systems (or between, if you must) is even easier.

Because development is such a large part of the Linux movement, people are very sensitive to all sorts of programming issues, including devout preferences for certain keystrokes over others and one editor over another. CodeWarrior is flexible in these areas, allowing (or requiring) third-party products such as editors and debuggers. However, some shortcomings are applicable to the open-source, multi-platform world of GNU/Linux.

Development Issues

One advantage of the CodeWarrior IDE is that it eliminates the complications involved in writing and maintaining a Makefile. Probably the main disadvantage is that it eliminates the advanced functionality and power that go along with the Makefile. This means that on one level, anything you write with CodeWarrior cannot be compiled on non-CodeWarrior systems unless you write your own Makefile to correspond with the CodeWarrior .mpc file (not a difficult task). The other side of this problem is that if you want to use CodeWarrior to hack some typical open-source software, you have to convert the Makefile first. Frankly, this is often quite difficult, and it defeats a large part of the purpose of an IDE—you might do just as well or better with Emacs/vi/joe/pico in one window and a terminal running make in another.

One possible solution would be for Metrowerks to add a Makefile importer *and* exporter. (Macintosh and Windows versions are said to have a Makefile importer, though I have not had a chance to use it.) If there were an open-source version of this converter for Linux, developers who use CodeWarrior

could easily cooperate with those who use other IDEs or the usual combination of Emacs (or vi) and make. A two-way converter may be a mildly difficult undertaking, considering the flexibility of make (with **configure**) compared to .mpc which is quite specialized. Still, it would be nice. So, although CodeWarrior may make cross-platform development much easier, it may slightly complicate cross-UNIX development. KDE developers in particular may be annoyed that it is difficult to use extended C++ class libraries like Qt. (I had trouble getting KDE programs to compile.)

Serious programmers may find other limitations. For example, third-party products (such as the debugger) are not integrated into the IDE, they are just loaded for you. Also, CodeWarrior does not support the Gnome/KDE drag-and-drop standards (who expects this?), and there seems to be no support for languages outside of C/C++ except as provided by **egcs** (for example, you can write in assembler). Also, if you would like to use KAI's C++ or any compiler other than gcc/egcs, the GNU Edition of CodeWarrior does not support it—this edition is based exclusively on free development tools. Now that we know what CodeWarrior is not, we can take a look at what CodeWarrior does have to offer.

Installation

Although installing the CodeWarrior IDE is quite simple (./install.sh from the root directory of the CD-ROM), getting a fully functioning system can be a bit complicated and involves some compilation. That is, if you want DDD (DataDisplayDebugger, a graphical front end for several debuggers including **gdb**), you have to get it from the CD-ROM, and if you want to do C++ development, you have to get egcs-1.1.2 from the CD-ROM, compile and install it (unless you have these things already). Also, you must edit your profile (or make symlinks) so that egcs will be used instead of gcc. Probably anyone fluent enough in Linux to develop projects, which need to be managed and integrated, can also unpack a .tgz file using **tar -xzf** and run **make**. Still, these things ought to have automatic installation available—ease of use is supposedly one of the benefits of commercial software. Since CodeWarrior currently exists for only Red Hat 5.2 and 6.0 (with SuSE soon to come), it should not be that difficult to make an easier, automatic installation program.

One element of installation which long-time free software users may find amusing is the license. When you start the install.sh script, you are greeted by several pages' worth of a “licensing agreement” which you must accept or the software won't install. The language is a bit pushy legalese, but somehow it's awfully funny to find something like this on Linux. (After all, who ever reads that stuff when Netscape first starts up?) I don't know if these last-minute licenses are legally binding, although you can supposedly return the software to the dealer and obtain a refund if you disagree. Either way, the software is more useful if you choose to install it.

Look and Feel

Figure 1. CodeWarrior Screenshot

Ideally, an integrated development environment should be a pleasant one. This means users should like the various menus, buttons, windows, bars, other functional interfaces, the debugger and most of all the editor (unless you spend more time in the debugger). CodeWarrior's editor is neat enough; it puts comments, keywords and strings in different colors than the main code, which is nice for some people. (Actually, it *would* be neat if console editors did this more easily.) In fact, you can customize all sorts of coloration into the editor. It does not, however, automatically parse and produce attractive C code the way Emacs does. Also, scrolling the page with the scroll bar or the PageUp/ PageDown keys leaves the cursor where it was, so once you find the correct line, it gets lost when you press a cursor key (you must click with the mouse to get your cursor to the right location). This could be good or bad, depending on whether you tend to scroll through the code and either forget where you began or forget which line you found.

As far as windows and menus are concerned, everything is laid out in logical places and the menu entries are diverse enough to offer a lot of varied functionality, while at the same time there are not more entries than there ought to be. The file menu is straightforward and simple, although the scroll bar does not scroll through files in real time; that is, it updates the screen when you've finished moving the bar. The Search menu has much to offer for searching, the Project menu contains a whole bunch of project-oriented functions (this is where you “make”), the Debug has many apparently permanently ghosted options (the debugger is not integrated with CodeWarrior —only loaded by it), there is a Window menu for dealing with the numerous windows of code, and an Info menu which actually has some configuration options for the IDE. All in all, it's actually fairly easy and obvious to get around and code. People who have never used CodeWarrior before may nevertheless find it familiar, and perhaps somewhat eerie, when the machine knows exactly what to do despite being given somewhat vague instructions.

Application Demo

As a short and clever method for introducing people to the CodeWarrior IDE, Metrowerks has included an Application Tutorial which is a version of GNUGO, edited to include a couple of bugs so that you can learn the debugger. The application tutorial goes through the steps of importing the source-code package into CodeWarrior, editing, compiling and debugging it. One curious quality of the 70-page manual in which these instructions are detailed is that often the details are not exactly correct. Still, it is not difficult to figure out what is actually meant (for example, select “Break at” instead of “Set Breakpoint”).

The “walk-through” is just that, plain and simple, yet adequate to introduce a user to the steps of program development. The learning curve is, well, short. From the moment CodeWarrior booted, I could intuit everything I would need to do. The names of functions are self-explanatory, the software never once crashed or froze (even temporarily), and the only thing that confused me was I had to change permissions on the source directory (chmod 777, actually) before I could get started. Also, I appreciate that Metrowerks chose GNUGO for the demo program because this excellent game is often overlooked.

For Red Hat Linux

At this time, CodeWarrior is available only for Red Hat Linux, which is bound to raise some eyebrows. What would happen to Linux if commercial vendors made their products available only to users of certain distributions? While it is true that distributors sometimes put important files in different places, normal software tends to compile across distributions and across hardware platforms, so it seems that CodeWarrior ought to work with any distribution. It should not be more difficult than making some symlinks or running make, so Metrowerks may have other reasons.

While the reason for choosing Red Hat seems clear (largest market share in the U.S.), the decision to support only one vendor was quite strange and seemed a bit scandalous. I hope no one switches distributions just for the ability to run CodeWarrior. However, the news on this front is that SuSE and Metrowerks have announced that they intend to produce and market CodeWarrior for SuSE. While it remains a mystery why CodeWarrior is so distribution-specific, the speculation is that Metrowerks cannot give tech support to so many distributions, and relies on the distributors to do so. It would be strange for a distributor to support someone else's product, and it *is* better to have an unsupported product than no product at all, but this may be the situation. Smaller distributors are at a disadvantage, so hopefully this practice will be abandoned before it becomes widespread.

Prognosis

Many people have been waiting for years to see CodeWarrior show up for Linux, and for these people and anyone who likes integrated development environments in general (very popular on Windows), this resembles what they have been wanting. In terms of functionality, it is not yet entirely comparable to CodeWarrior for Macintosh or Windows, and limitations to the GNU Edition are present. However, many coders are excited to find out what future releases and the Professional Edition will have in store. You may also want to take a look at the open-source Code Crusader and Cygnus Code Fusion. Software development firms could become especially fond of an IDE like CodeWarrior, and if this problem of Makefile vs .mpc is resolved, it will become a viable

option for individual, at-home developers who want to contribute to the GNU/Linux world of open-source software.

In any event, we can hope that CodeWarrior's presence on our platform will make programming easier, thereby inspiring more programmers. The world of GNU/Linux is extremely different from that of Macintosh and Windows, and CodeWarrior will need to make many adaptations to be successful. Hopefully, this product is first in a line of efforts to bring the immensely popular CodeWarrior IDE to Linux.

Jason Kroll still thinks GNU/Linux is the best thing to happen to computers since monitors. He can be reached at info@linuxjournal.com.

Archive Index Issue Table of Contents

Advanced search

NexStar

Jason Kroll

Issue #66, October 1999

The true question, aside from the color, is how well the Linux system operates.

- Manufacturer: NexTrend Technology, Inc.
- E-mail: sales@nextrendpc.com/
- URL: www.nextrendpc.com
- Price: $1419 US with 17" Monitor $1139 US without
- Reviewer: Jason Kroll

Successful PC manufacturer NexTrend has begun offering complete, pre-installed Linux systems based on Caldera OpenLinux 2.2. The systems arrive completely configured for all hardware and even the X Window System, and are available in a variety of configurations based on different processors and hardware components. The systems include everything one expects in a computer, including keyboard, mouse, shielded speakers, the computer itself and a bunch of mysterious manuals, screws and bubble wrap (tension sheets?)--the typical things that come with computers, some of which we never figure out how to use (for example, the free Windows software that comes with your Linux box).

The real question is, what's the very best thing about this computer? It's turquoise. Actually, the particular machine NexTrend sent to our product testing lab has a Pentium III 500MHz CPU, an 8.4GB drive and 128MB of RAM,

which are also nice. To be honest, this machine is actually quite loaded. Still, if you order one, you can have more or less any configuration, as far as I know. The true question, aside from the color, is how well the Linux system operates.

It looks like a Macintosh, but we all knew that about turquoise-colored machines running KDE. Still, what with the color of the machine (perhaps more accurately described as teal), it feels particularly inviting, on top of which Caldera OpenLinux 2.2 is a tight, clean distribution which includes a lot of productivity software preconfigured into the menuing system (such as Netscape, WordPerfect 8.0 and Star Office 5). COAS (Caldera Open Administration System) is a small, fast program for dealing with things ranging from system administration to machine-specific hardware configuration, and even includes a loader for kernel modules. Taken on the whole, Caldera OpenLinux 2.2 fits very nicely on the home desktop computer, and while it is known as “Linux for Business”, the important point is it works, is easy to use and comes with a lot of software, though still no **pico**.

Although Caldera has about the easiest installer going, one way around the alleged difficulty of Linux installation is to buy a pre-installed system. The NexStar system is completely configured, with an optimal kernel and dozens of modules. Everything works on this machine, from the XF86Config file, to the Sound Blaster 64AWE card, the 56K modem, the Intel 8460B Ethernet card, the 48X speed CD-ROM drive, the Iomega ZIP drive (though this is not pre-configured into fstab), the floppy drive, the 8.4GB hard drive (okay, if this didn't work, there'd be a problem), the ATI AGP 8MB video card, etc. XF86Config is ideally configured for the optional CyberVision monitor in both 800x600 (84Hz) and 1024x768 (70Hz) modes (I added a nice 1152x864 mode at 69Hz), though libvga.config is not configured (just copy data from XF86Config). The video timings are more or less perfectly matched to the monitor, and when the system is up and running, it's quite neat. However, it *is* possible to make a mess of things.

The one annoyance in OpenLinux 2.2 is that once you have “shutdown” KDE to console mode, you cannot get it up again. **startx** and **kde** just start up a bare X screen and the client has authority issues connecting to the server, so all you get to do is move an X over a test pattern. Typing **telinit 5** won't solve this problem, but you can reboot or else just remember not to shut down KDE. The remedy, an update of XFree86, is available on Caldera's web site. OpenLinux 2.2 has the feel one expects of a professional Linux system, and COAS is truly neat. While hackers often like Slackware, and SuSE, Debian GNU/Linux, and the ubiquitous Red Hat are also superb choices, Caldera OpenLinux 2.2 is ideal for the user who wants a pre-installed system—it has a *lot* of software organized neatly into KDE's menus, is a complete system and truly feels like one.

NexTrend's selection of OpenLinux is a point to consider when deciding which Linux PC maker to choose, if for example you're bored with Red Hat.

No computer review is complete without some benchmarks, but we haven't finished our custom benchmark software, so it's time for benchmark trivia. The **cat /proc/cpuinfo** command reports 498.07 bogomips; the Crafty chess engine evaluates about 190,000 positions per second; **bonnie** reports block writing at 21533KB/sec and block input at 14684KB/sec, and BYTEmark evaluates the system to have a memory index of 2.082, an integer index of 1.794, and a floating-point index of 4.603 (compared to AMDK6/233). It took 3.5 minutes to **make bzImage** for a 462KB kernel.

Many machines are out there, so be sure to shop around, and if you order from NexTrend, make sure they still put Linux boxes in these cool, teal shells.

Jason Kroll is your amiable Technical Editor who frequently reviews hardware and software, and is rumored to have interests outside of computers. He can be reached at info@linuxjournal.com.

Archive Index Issue Table of Contents

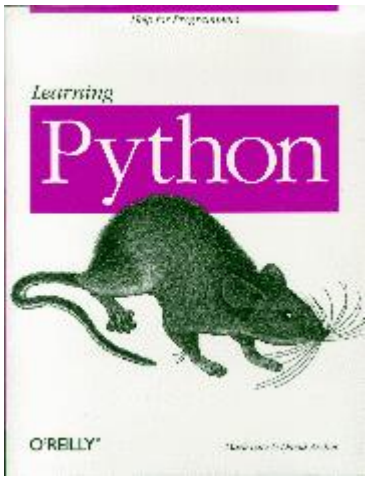
Advanced search

Learning Python

Phil Hughes

Issue #66, October 1999

I see Python as the interpreted language for those who expect to be able to go back and understand their program a year after they have written it.



- Authors: Mark Lutz and David Ascher
- Publisher: O'Reilly & Associates
- E-mail: info@ora.com
- URL: http://www.ora.com/
- Price: $29.95 US
- ISBN: 1-56592-464-9
- Reviewer: Phil Hughes

To understand what this review means, you will need to know about two things: what Python is and my background. Python is an object-oriented, interpreted programming language suitable for scripting tasks as well as serious programming projects. I see Python as the interpreted language for those who expect to be able to go back and understand their program a year after they have written it.

Now, about me. You may know me as a magazine publisher, but I am truly just a geek who found a way to make less money. I have been writing in assembly language and FORTRAN since the '60s, and in more UNIX-like languages such as C and AWK since 1980. I have seriously used at least a dozen languages and am generally very comfortable around most anything except Cobol.

What I am not is an object-oriented programmer. I understand the concepts, but have never worked in an object-oriented language such as C++ or Modula.

Python is interpreted like Perl or awk, but it is object-oriented. I was ready to give it a try. The problem is that I am not a full-time programmer, because I have this publishing job to do, but many times I do end up writing code.

Armed with O'Reilly's *Programming Python*, I was off to become a Python expert. Well, to make a short story shorter, it didn't work. While it is a good book to use as a reference or to take with you to a desert isle along with your Python-equipped laptop, it wasn't the book for a part-time programmer with over 30 years of non-OOP experience to use.

Enter *Learning Python*. My executive summary is that this is the right book for me and probably for many others as well. While *Learning Python* doesn't tell you everything, it is a good 366 pages that will get you up and running. Written in a textbook style with examples and exercises, it introduces both object-oriented programming and the Python language.

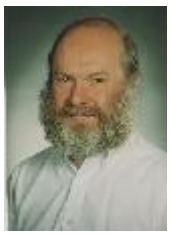
Both authors have done Python training, and it shows. Examples appear where you need them, and the exercises actually test your understanding of important concepts. This is a book to read with a computer nearby. You will learn a great deal from the exercises.

The book is divided into three parts. The first part covers the core of the Python language, explaining types and operators, basic statements, functions, modules, classes and exceptions. Part two moves you out a little into Python's built-in tools, common tasks and finally how to build real programs. Part three covers Python resources on the Net, platform specifics and answers to the exercises.

I have been very thorough in going through this book (because I actually want to add Python to my language set), and have found the book to be extremely accurate. The examples all work and you won't be misled by the text—a problem far too common for first printings of technical books.

Who, besides me, should get this book? I would say anyone who is comfortable with computers and wants to learn a very cool object-oriented language. By

“comfortable”, knowing one programming language or at least a scripting language is going to help a lot. While the book covers the basics, if expressions like “dynamic typing” or “syntax rules” scare you, then you may need to get a little more comfortable before attempting to learn a real programming language.



Phil Hughes is the publisher of *Linux Journal*. He can be reached via e-mail at info@linuxjournal.com.

Archive Index Issue Table of Contents

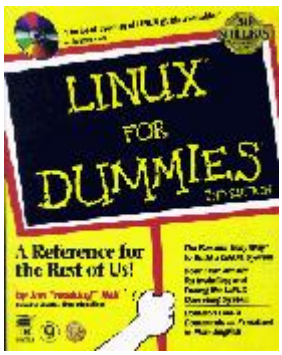
Advanced search

LINUX for Dummies 2nd Edition

Harvey Friedman

Issue #66, October 1999

Other than far too many typos, it appears to be a success.



- Author: Jon “maddog” Hall
- Publisher: IDG Books Worldwide, Inc.
- E-mail: siteemail@idgbooks.com
- URL: http://www.idgbooks.com/
- Price: $24.99 US
- ISBN: 0-7645-0421-5
- Reviewer: Harvey Friedman

The 2nd edition of *Linux for Dummies* shares little more than a name with the 1st edition and is, I think, vastly improved. The apparent focus is an introduction to Linux for a user of Microsoft Windows, with explicit instructions and many screen snapshots. Other than far too many typos, it appears to be a success. The book contains a CD-ROM with Red Hat Linux 5.2 for an Intel X86 PC, and a tear-out “cheat sheet” containing names and brief descriptions of some useful commands. The UNIX novice will probably use it to look at the description, then read the man page for the command syntax.

The first 30 pages introduce Linux, give a brief history and describe some of what an experienced user can do with it—all designed to convince a possibly reluctant person to try it.

The next 90 pages are what the average reader would expect from a "... for Dummies" book—an MS Windows user is led step-by-step in installing Linux, after first determining whether the computer is comprised of hardware on which Linux can run. Assuming the user is naïve regarding computers, Hall describes how to use MS Windows, including screen dumps as illustrations, to determine what hardware is in the computer system. So, in many cases, the user learns more about how to use a familiar OS in his attempt to install Linux. This builds confidence and is a good instructional technique.

Next, writing boot and supplemental floppies from the included CD using rawrite.exe, defragmenting, repartitioning with FIPS and **fdisk** are covered. Naturally, Hall recommends using a separate disk for Linux so that repartitioning for Linux is not necessary. Many screen dumps are provided for the installation procedure, including both Disk Druid and fdisk. Configuring a mouse, the X Window System, networking, setting the system clock, setting boot services, choosing a root password, creating a boot floppy disk to boot the system one has just installed, and configuring the bootstrap loader, LILO, are described. Because the X Window System is so important to the rest of the development, a chapter is included on "Solving Problems with the X Window System Installation". Adding a user through the control panel in X is shown, because that is the most likely method of choice for MS Windows users. Hall does include a page on adding users from the command line, and I have to admit I was surprised to see the command for that was not **adduser** but rather **useradd**. Then I realized this change parallels the terminology of the X Window User Configurator. Awkward, but it works!

The next 100 pages introduce files, commands, editors (mostly **vi**), shells, using X, calendar, calculator and configuring a sound card. As of this writing, I have not yet gotten my old MediaVision Deluxe JAZZ16 sound card to work. However, I must admit I've spent more time on this review than on such a frivolity.

Maintenance is the topic of the next 30 pages: managing the file system, including mounting and dismounting, adding more disk storage and using **mtools** to work with MS-DOS floppies (read, write, etc.). Using CD-ROMs in Linux is discussed in one chapter and tuning the system and building the kernel in another.

The next chapter illustrates how to set up a modem and a PPP connection to one's ISP via X. In particular, one need not worry about writing involved scripts for PPP as long as one knows the phone number, the IP address(es) of

nameserver(s) and can respond to the dialog and screen dumps in the book. I tried only PPP, but the dialog box offers the choice of SLIP, PLIP, Token Ring and others as well.

Since one probably wants the PPP connection in order to surf the Web, the next chapter is all about setting up and using Netscape. This, too, works nicely using **ifup** in an xterm.

The obligatory (in every "...for Dummies" book) "Part of Tens" is next. One chapter presents ten sources of help, which should offer something for everyone; another, ten problem areas and solutions, usually referred to as FAQs.

Appendix A is a list of hardware compatible with Linux, B discusses man pages, and C is a brief description of the CD-ROM.

I think Hall did a tremendous job of completely rewriting the book, but wish he could have had better support, particularly a technical copy editor. There are just too many typos (or thinkos) in this book for the leery first-time user to feel totally confident. I have not had time to find all the problems, but two follow. One, on page 52, "formed" is used when referring to a 1.44 MB floppy disk; it is clear the word should have been "formatted". Two, a more serious problem occurs on page 309 in the problem and solution chapter. When the **ls** command doesn't show files in color, a solution is offered that is correct in principle, but wrong in detail. Adding the line

```
alias ls ='ls - color=auto'
```

to the .bashrc file will not change ls to use color for the different file types. It is not clear the new user would know enough to type **man ls** to find the correct syntax to solve the problem.

```
alias ls ='ls --color=auto'
```

will work; in fact, leaving off **=auto** will also work for the included Red Hat 5.2 CD-ROM version.

I can recommend this book for the person who works with Microsoft products but wants to try Linux. Despite too many minor errors and inaccuracies, following the directions in the book will result in an installed Linux, X and Net connection, *assuming* one's hardware is supported (check the appendix) and one knows the phone number, name servers and gateway for the ISP.

Overall, while this is a usable book for installing Linux, the user should also have *Linux for Dummies, Quick Reference, 2nd Edition* by Phil Hughes, to learn the more common command-line interface instructions with examples.

I liked this book and hope the next edition will clean up all the typos and maybe even add a chapter on installing DOSEMU and Wine. Unlike the first edition, the second is a book that leads the reader in a step-by-step fashion toward success.

Harvey Friedman is a computer consultant at the University of Washington. He can be reached via e-mail at fnharvey@u.washington.edu.

Archive Index Issue Table of Contents

Advanced search

Creating CDs

Alex Withers

Issue #66, October 1999

Complete instructions for storing your data on CD.

These days, everyone seems to have a CD writer (CD-R). They are great pieces of hardware, and prices are going down all the time. CDs are great for keeping static data. Your favorite downloads contained on piles of floppies can now be transferred to one CD, lowering your risk of losing the data. With your own CD writer, you can make custom CDs of your favorite Linux distribution, crafted for your needs. You can also make CD archives of your favorite FTP site and pass it along to friends to avoid bandwidth problems.

Choosing and Setting Up Your CD Writer

If you are currently looking for a CD-R, get SCSI. SCSI CD-Rs have been around far longer than ATAPI (IDE) CD-Rs and, as a result, are better supported. However, if you do have an ATAPI CD-R or just can't afford SCSI, don't worry—you can still make CDs.

In order to write with a SCSI CD-R, you must have “SCSI support”, “SCSI CD-ROM support” and “SCSI generic support” compiled in the kernel. Also be sure you have “ISO9660 cdrom filesystem” support. Having “Loopback device support” compiled in the kernel is a good idea, but not required. For this article, I will assume you know how to build your own kernel; if not, refer to the Kernel-HOWTO (see Resources).

ATAPI CD-Rs require a bit more effort. You must have at least kernel version 2.0.35; anything below this requires patches. Recompile your kernel with “IDE/ATAPI CD-ROM support” disabled and “SCSI Emulation support” enabled. Along with these options, you must also enable those options mentioned above (yes, even the SCSI support). The result is your CD-R will look and act like a SCSI device, even though it is an ATAPI. Your CD-writing software also needs to support ATAPI writing; I will discuss this later. Note that when you use SCSI

emulation, all IDE CD-ROMs change to a SCSI prefix, so your first CD-ROM would be /dev/scd0.

Required Tools

CD writing in Linux requires two utilities: **mkisofs** and **cdrecord**. The first, mkisofs, is required to make an image of the files you wish to burn. Most major distributions come with this utility, but in any case, getting the latest version would be wise. The latest version can be found at ftp://tsx-11.mit.edu/pub/linux/packages/mkisofs/. The second utility, cdrecord, is the software used to burn the image made with mkisofs to a blank CD. You can find cdrecord at metalab.unc.edu/pub/Linux/utils/disk-management, and again, I highly recommend getting the latest version (1.6.1 at the time of this writing). Another utility for burning CDs is **cdwrite** which can also be found at that URL.

Aside from these, some free and commercial GUI programs for making CDs are also available. X-CD-Roast (see "X-CD-Roast: CD Writer Software" by Thomas Niederreiter, *LJ*, January 1998) is probably the most famous and is freely available; however, it is based upon the cdwrite utility. There are also X applications which act as a shell and call upon the appropriate utilities to do the job.

Collecting the Files

Once you have the appropriate utilities installed, you can begin to write your own CDs. Before you begin, collect the files to be burned under one directory. I'm going to use the downloading and burning of a Linux distribution as an example throughout this article. First, let's say you anonymously log in to ftp.*some_server*.com and switch to the directory /pub/linux. This hypothetical directory contains three different distributions; the list output **ls** looks like Listing 1.

Listing 1. Directory Listing

Assuming distribution_b is the one you want, proceed to download the entire distribution by typing **get distribution_b.tar** at the ftp prompt. Of course, this directory contains everything you need and nothing else. After all, you wouldn't want to download the distribution for several different architectures. Thus, when you download an entire distribution, make sure you go deep enough into the directory to get only what you need. For example, if you wanted to download the entire Red Hat distribution for the i386 architecture on ftp.cdrom.com, you would download everything in the /pub/linux/redhat/redhat-6.0/i386 directory and below.

Once the download is finished, create a directory for storing the files. In this example, I'll create a directory called cdimage, then place the contents of distribution_b.tar into that directory by executing the following commands:

```
mkdir cdimage
tar vxf distribution_b.tar -C cdimage
```

Since the verbose (**v**) option is requested, the contents of distribution_b.tar will fly by on your screen and its contents will be located in cdimage. If everything went well, you can now delete distribution_b.tar to save space.

If you want to burn some collection of files other than a Linux distribution, just place them in the cdimage directory.

Creating the CD Image

Remember when creating CDs, the root of the CD is relative to the created directory; in this case, cdimage. Once you have your files in this directory, you are ready to create the iso9660 image using the mkisofs command. To create the basic image, use the following command:

```
mkisofs -r -o cdimage.iso cdimage
```

The **-r** option ensures the image contains additional file description data by way of the Rock Ridge protocol, preserving the original file name and setting permissions optimally for CD-ROMs such that read/execute permissions become global, write permissions are cleared, and special mode bits are also cleared since they do not apply on CD-ROMs. The **-o** option designates the output file (cdimage.iso). The last value is the directory in which the files are located.

Many commercially manufactured Linux CDs, such as Red Hat, are bootable. This isn't difficult to do using the “El Torito” standard. Most newer BIOSes today support the bootable CD feature, and most bootable CDs for the PC are based on El Torito. El Torito makes your CD appear as a floppy, and thus your BIOS can boot it.

If you want a bootable CD, you'll need a 1.44MB boot image intended for a boot floppy. In our distribution example, we could use the boot image used for installation. For distribution_b, the name of the boot image is boot.img. The process by which we make a CD bootable takes place in the creation of the iso9660 image (International Organization for Standardization specification for compact disk read-only memory). Thus, before we create our image, we need to create a directory inside of cdimage to hold the boot image; a directory called boot would work fine. So, we place the image boot.img into cdimage/boot and create the iso9660 image by executing the following command:

```
mkisofs -r -b boot/boot.img -c boot/boot.cat -o\
   cdimage.iso cdimage
```

Here we have two new options, both of which are used to make the CD bootable. The **-b** option is followed by the name of the boot image to be booted. Note that the file is relative to the root of the CD. The **-c** option is followed by the name of the boot catalog required by El Torito; this file is automatically created by mkisofs. Only the more recent versions of mkisofs allow for the automatic creation of the boot catalog; older versions require you to create it yourself.

Before actually burning the CD, take a look at your image layout by mounting it. This is done using a loopback device, so this must be supported in the kernel. The following command will mount your image:

```
mount -r -t iso9660 -o loop cdimage.iso /mnt
```

Burning the CD

Once you've created your image, bootable or not, you are ready for the final process of burning it onto the CD using either cdrecord or cdwrite. Take the following into consideration before you start:

- Make sure the computer isn't experiencing any excessive vibrations.
- Make sure the image is on a local hard drive.
- Make sure the load on your system isn't too high.

Keeping these three things in mind will help prevent errors during the write process. The CD writer can be put through a test process that won't actually write, but will simulate the entire process. This is done by adding the **-dummy** option for cdrecord and the **-y** option for cdwrite. Now all that is left is inserting a blank CD and executing the command that matches your choice of writer.

```
cdrecord -eject -v -isosize speed=2 dev=0,0\
   cdimage.iso
cdwrite -ev --device /dev/??? -s 2 cdimage.iso
```

The first two options for both utilities are eject and verbose. Thus, the CD will eject after the burning process is finished, and the program will run in verbose mode. The option **-isosize** for cdrecord limits the size of the CD to the size of the iso9660 image. The options **speed=2** and **-s 2** indicate the speed at which to write to the CD; in this case, the **2** means at 2x. Finally, the options **--device, dev=0,0** and **/dev/???** set the target device, where **/dev/???** should be your CD-R (i.e., /dev/scd0) and **0,0** stands for the SCSI ID and bus in that order.

Conclusion

If all worked out well, congratulations. You now have a full-fledged, iso9660 CD. You can make a CD with other formats just as well. The utility **mkhybrid**, included with mkisofs, can make images of Joliet and HFS format. It is also possible to make a CD with the EXT2 file system format. Have fun burning!

Resources

Alex Withers has been using Linux since 1.1.59. He is currently studying computer science at Gonzaga University and can be reached at awithers@gonzaga.edu.

Archive Index Issue Table of Contents

Advanced search

LINUX JOURNAL

Dynamic Graphics and Personalization

Reuven M. Lerner

Issue #66, October 1999

A continuation of the discussion on creating graphics dynamically on the Web.

Last month, we discussed the different ways a CGI program can create dynamically generated graphics output. That is, we wrote several programs in which the program describes its output not as “text/html”, but as “graphics/gif”.

This month, we will examine some more tools that allow us to create graphics dynamically. However, the graphics will have an additional twist this time, in that they will reflect an individual user's stock portfolio rather than a global set of data values.

As with any non-trivial software project, our first step must be to create a brief specification. In this particular project, we will have two major programs. In the first, the user will be able to create and edit a personal profile, describing the securities he or she owns. The second program will take the information in the user's profile and use it to create a personalized graphic stock portfolio.

Needed Tools

This project brings together a number of tools we have discussed in previous installments of ATF. Nevertheless, it seems like a good idea for us to review them, since we are going to call on so many.

MySQL: MySQL is a small, inexpensive relational database available for Linux and many other operating systems. (See “Resources” for information on where to get it.) In addition to its low price, MySQL is quite fast and efficient, which makes it popular on many web sites. As a relational database, MySQL forces us to store information in one or more tables, in which each row refers to a separate record. As with most relational databases, we communicate with MySQL using SQL, a database query language. We cannot write programs in

SQL; rather, we must embed our queries inside of a program written in a full programming language. In our case, that language is Perl.

DBI: While SQL might be a standard query language for communicating with databases, the software and libraries used to speak with those databases vary considerably. To talk to an Oracle server, you need Oracle libraries; to talk to a MySQL server, you need MySQL libraries, and so forth. As a result, the Perl database world was fractured for a long time, with special versions for individual databases.

Now, however, there is a better way: the DBI module standardized the API to relational databases, meaning that programmers moving from one database server to another have to learn only the different nuances of the SQL implementations. Previously, they had to learn a separate Perl API as well, which was frustrating. This was accomplished by separating the database code into two parts, one generic (DBI) and the other specific to each server (DBD). In order to use DBI, you will need to install the generic DBI libraries, and then one or more DBDs appropriate for the products you use.

There is a problem with DBI and the Web, however, which has to do with the way in which database servers were designed. In general, they expect a client program to open a connection, perform many queries, then disconnect. Opening a connection is thus quite slow and inefficient. When a CGI program is a database client, it must open a new database connection for each HTTP transaction. See the mod_perl section immediately below for one solution to this problem, Apache::DBI.

mod_perl for Apache: Web servers traditionally provided custom and dynamic output by invoking external programs, using the CGI standard. An HTTP server would pass information to the CGI program, which would then be expected to send its output to the user's browser. This output generally came in the form of HTML-formatted text, but as we saw last month, it is possible to produce graphics as well.

However, CGI is quite slow; every invocation of a CGI program requires a new process to be created. If you are using Perl, each invocation requires the program to be compiled into Perl's internal format, then executed.

An alternative method is to use mod_perl, a module for the free Apache HTTP server that embeds a fully working version of Perl inside the server. This has several ramifications, one of which being the fact that we can now create custom output without having to rely on external programs.

When using mod_perl, you can take advantage of a module known as **Apache::DBI**. This module pretends to work the same as DBI, but actually caches database handles ($dbh) across invocations. So even when your program thinks it is opening a new database connection, it is actually reusing a database handle from a previous invocation.

GIFgraph: The GIFgraph set of Perl modules allows us to create charts and graphs on the fly, from within CGI programs or mod_perl modules. We explored a basic use of GIFgraph last month. As its name implies, GIFgraph produces output in GIF format. Last month, we saw how to return GIFs directly to the user's browser. This month, we will instead save the resulting graphs to individual files, to which we will create hyperlinks.

Apache::Session: HTTP, the protocol on which the Web is based, was designed to be lightweight and simple. As part of this consideration, it was also designed to be “stateless”, meaning each transaction is independent. This creates a problem, however, in that you will often want to keep track of which user is which. For instance, in this application, we want to ensure we are tracking the correct user's portfolio. Without state, we cannot track portfolios at all, let alone for multiple users. Apache::Session, as we will see below, allows us to get around this by using a database and HTTP cookies to store one or more pieces of information using a unique identifier.

With the above five technologies, we can create a fairly impressive stock portfolio tracker that allows users to define which securities they own and view their current holdings in graphical format. As presented this month, the application is admittedly a bit crude, but it should show you how easy it is to write such an application and how flexible the above tools can be in creating one.

Creating the Database Tables

Before we can begin to work on the applications themselves, we need to create the underlying database tables they will use. We will need two different tables: one to hold the individual stock values on different dates and another to store user personalization information.

The first table, called StockValues, has three columns: a symbol, which can contain up to six characters; a value, which can range from 0 to 999999.999; and a date. We can create such a table with the following SQL, most often by using the interactive **mysql** client program that comes with MySQL:

```
CREATE TABLE StockValues (
   symbol CHAR(6) NOT NULL,
   value NUMERIC(6,3) NOT NULL,
   date DATE NOT NULL
);
```

Each row in the above table refers to the value of a single stock on a single day. By storing information like this, we can easily create charts for a stock during arbitrary periods of time. For the sake of brevity, our applications will always display all of the available values for a stock. The above table also gives us many possibilities for additional applications, such as finding a stock's high and low values during a given time period.

How will StockValues be populated with values? Most commercial sites retrieve stock information from a commercial service, using a background process to place the information in a database table. My budget is more limited than the average business web site, so I decided to insert some arbitrary values into StockValues. In order to do this, I used the interactive mysql client program, and entered several queries of this type:

```
INSERT INTO StockValues (symbol, value, date)
   VALUES ("ZZZZ", 100, "1999-07-14");
```

The second table we will create is for Apache::Session::DBI, a version of Apache::Session that allows us to store information about a particular user in a database table. The table's name and format are determined by the Apache::Session API:

```
CREATE TABLE sessions (
   id char(16),
   length int(11),
   a_session text
);
```

Once we have created this table, we can ignore the fact that Apache::Session stores its information in a database. So far as we are concerned, we perform a magic incantation at the beginning of our code, which retrieves the current session values. We retrieve the user's session ID by reading an HTTP cookie:

```
my $id = $r->header_in('Cookie');
$id =~ s|SESSION_ID=(\w*)|$1|;
```

Then, once we have assigned **$id** the value of the user's session ID, we tie the **%session** hash to the “sessions” table with the Apache::Session::DBI module:

```
my %session;
tie %session, 'Apache::Session::DBI', $id,
{
DataSource => 'dbi:mysql:test:localhost:3306',
UserName   => '',
Password   => ''
};
```

From this point on, any name,value pairs stored in **%session** in previous sessions will be available. By the same token, we can assign

```
$session{key} = "value";
```

and be sure that in our next invocation, despite HTTP's statelessness, we can retrieve the same value. Apache::Session thus makes it possible for us to store arbitrary quantities and types of information about a user.

We will store three session variables for each user. The e-mail address and name will be stored as scalars, and the user's current holdings will be stored as a hash reference. The keys to the **%portfolio** hash will be the stock symbols, and the number of shares owned in that particular security will be stored as the values.

When we want to store **%portfolio** as part of the session, we turn it into a reference and store that in **%session** with the key “portfolio”:

```
$session{portfolio} = \%portfolio;
```

A reference is a specially tagged scalar, which allows us to store it in a hash. We retrieve it later with the following complicated-looking code:

```
my %portfolio =
   defined $session{portfolio} ?
   %{$session{portfolio}} : ();
```

The above uses Perl's trinary operator **?:** as a shortcut to “if-then”. It means that if $session{portfolio} is defined, then dereference it into its original hash value and assign it to **%portfolio**. If it is undefined, then assign the empty hash, **()**, to **%portfolio**. After this line of code is executed, **%portfolio** will contain the user's current portfolio. By using Apache::Session, we can maintain the illusion of state across HTTP transactions, and store many users' portfolios in our database.

Profile Editor

Now we will write the two applications that will work with this information. The code for those two applications can be found in the archive file at ftp://ftp.linuxjournal.com/pub/lj/listings/issue66/3629.tgz. The first will be StockProfile.pm, a Perl module for mod_perl that will allow users to create and edit their portfolios and personal information.

Since our program will be running as part of mod_perl, we will need to remember several things. First and foremost, we must create a new Perl module and package with a subroutine named “handler”. We will configure Apache to invoke this “handler” subroutine whenever a particular URL is requested from the HTTP server. Because our subroutine will be part of Apache rather than invoked in a separate process, and because mod_perl compiles and caches code that we write, our routine will run much faster than if it were a CGI program.

We must also remember to adhere to mod_perl programming conventions, the most important being to use lexical (“temporary” or “my”) variables as much as possible. Global variables stick around across invocations of mod_perl, which can lead to memory leaks and odd bugs. We ensure that we use “my” before variables with the **use strict** pragma at the top of our program.

Our module, Apache::StockProfile.pm (see Listing 1 in the archive file), has three stages: First, it initializes all of the variables and information, grabbing the current list of securities from **StockValues** and initializing the user's profile. Then, if the module was invoked with the **POST** method, it sets or modifies the user's profile information as necessary. Finally, it produces an HTML form that can be used to modify the profile further.

The first thing we do in “handler”, as with all mod_perl modules, is retrieve the Apache request object, traditionally known as **$r**. This object's methods allow us to retrieve and set everything having to do with the HTTP transaction. For instance, we can set outgoing headers with **$r->header_out**, the “Content-Type” header with **$r->content_type**, and send the final headers with **$r->send_http_header**.

However, certain things are more easily accomplished—at least to experienced CGI programmers—with CGI.pm, the standard module for CGI programming. We can get a version of that API by using and creating an instance of CGI::Apache. The created object gives us access to HTML form elements and debugging tools (including the invaluable **dump** method) using the familiar interface from CGI.pm. Not everything works in the same way, but it is good enough for almost all purposes.

Our main use of CGI::Apache in this program is to retrieve the HTML form elements, which are submitted via the POST method. StockProfile.pm both creates the form and handles its submission, which might seem odd at first but makes for a compact and easy-to-maintain type of code.

We retrieve a list of current symbols with a simple SELECT statement to the database. However, if we were to simply say “SELECT symbol FROM StockValues”, we would get one row for each value of a symbol or about five values per week, if we add one new value each day. In order to retrieve distinct values, we add the qualifier **DISTINCT** to our SELECT query. We also ask that the symbols be alphabetized, so that they will be listed in reasonable order:

```
my $sql = "SELECT DISTINCT symbol FROM StockValues";
$sql .=   "ORDER BY symbol ";
```

We set the values of **$name**, **$email** and **%portfolio** based on the user's session information, as we discussed earlier. We then use this information to fill in the

HTML form that allows the user to modify his or her profile. I prefer to use a table for such forms to ensure the columns line up straight, but that is simply a matter of aesthetics; the important thing is that each element must have its own, unique name and they will be the same names as our POST-handling code at the top of the program expects to use.

Stock Reporter

Our second mod_perl handler is StockReport.pm (see Listing 2 in the archive). This module uses the portfolio information the user has entered, and creates one or more graphs based on it.

If the user has a portfolio defined, then we iterate through each of the symbols in it. We then SELECT all of the rows in StockValues with that symbol, retrieving them in date order:

```
my $sql = "SELECT value,date FROM StockValues ";
$sql .=   "WHERE symbol = \"$symbol\" ";
$sql .=   "ORDER BY date ";
```

Now we iterate through each of the returned rows, adding the value to the **@values** array and the date to the **@dates** array. We also calculate the value of the user's holdings on that day (by multiplying the number of shares by the share price) and put that in the **@holdings** array.

We then plot our data set by creating a **@data** array, in which the elements are references to **@dates**, **@values** and **@holdings**. **@dates** will be used as the X axis in our graph, while **@values** and **@holdings** will be plotted as well. Because each element of **@holdings** is bound to be a multiple of its counterpart in **@values**, we tell GIFgraph to use two Y axes—one for values and one for holdings.

We create the graphs themselves using GIFgraph's **plot_to_gif** method, which takes a set of data points (the **@data** array, StockReport.pm), creates a graph in GIF format, then saves it to disk. We set the filename in a variable, so that we can both save the file and refer to it in an IMG tag. Remember, the file must be in the web document tree in order for it to be available to the user's web browser!

It might be tempting to put such files in /tmp, the standard temporary directory for Linux systems, but then the graphics will be unavailable to outside browsers. This directory must be writable by the web server, which often means making it open to more people than the rest of your web hierarchy. If this is the case on your system, make sure only this directory is writable by others, so that you don't run the risk of an intruder viewing or damaging your site's sensitive files.

Creating files in this way works well, but with one major flaw: it has the potential to fill your file system with numerous old graphics. A number of methods can be used to overcome this problem, but perhaps the simplest is to use **cron** to identify and delete any file older than a certain time. Depending on how busy your site is, you might want to run such a cron job every ten minutes, every hour, or once a month. It all depends on how many visitors you receive and how large your disk is. It is probably better to run such a deleting program more often, so as to avoid a denial-of-service attack that could fill your disks.

While I did not implement it in this version of StockReport, you can probably see how easy it would be to allow users to choose the range of dates in the graph. Using an HTML form, you could allow users to choose the starting and ending dates; the values of those form elements could then be inserted into the SQL query so as to SELECT just those rows between the named dates.

Installing the Modules

Once we have written and installed StockProfile.pm and StockReport.pm into our Perl module hierarchy, we must somehow tell Apache when to use them. We can do this in a number of ways, but my preference is to create special URLs that invoke these modules. That is, every time someone requests the URL "/stock-profile" from our server, they should get the profile editor. By the same token, when someone requests "/stock-report" from our server, they should see a report of their current stocks.

In order to accomplish this, we must first load each of the modules by adding the following two lines to the Apache configuration file, httpd.conf:

```
PerlModule Apache::StockProfile
PerlModule Apache::StockReport
```

Once we have done that, we can create new URLs, which do not necessarily correspond with files on the server's file system. For this, we use **<Location>** sections in httpd.conf. We indicate that the URL in question should be handled by a Perl module ("SetHandler perl-script"), and then tell Apache which specific module to use for that particular URL:

```
<Location /stock-profile>
   SetHandler perl-script
   PerlHandler Apache::StockProfile
</Location>
<Location /stock-report>
   SetHandler perl-script
   PerlHandler Apache::StockReport
</Location>
```

You will need to restart Apache in order for these new URLs to work. If there is an error in one of the Perl modules, or if mod_perl cannot find one of the modules in the module path **@INC**, the restart of Apache will fail. This ensures

you will not have any compile-time errors when your modules run under mod_perl. At the same time, it requires that you test your modules extensively before including them on a live site, since bringing down the server on a large web site can be embarrassing or financially difficult.

Conclusion

Last month, we saw how to create simple dynamic stock graphs using the GIFgraph package. This month, we saw how such dynamically created graphics can fit into a larger application, allowing users to see information about their stock portfolios. The combination of a relational database, mod_perl and GIFgraph makes it possible to create such a simple application in under 400 lines of code. You can undoubtedly think of many other applications in which dynamically created graphics would be useful—let your imagination go wild!

Resources

Reuven M. Lerner is an Internet and Web consultant living in Haifa, Israel. His book Core Perl will be published by Prentice-Hall later this year. Reuven can be reached at reuven@lerner.co.il. The ATF home page is at http://www.lerner.co.il/atf/.

Archive Index Issue Table of Contents

Advanced search

Linux in Kuala Lumpur

Dr.. Junaid Ahmed Zubairi

Issue #66, October 1999

Setting up computing facilities at a Maylasian university was easy using Linux.

My involvement with Linux began in 1994 when a friend of mine bought the complete set of Slackware diskettes for me. At that time, I was setting up computing facilities in Sir Syed University, Pakistan. Linux was set up as the first e-mail server on campus and was later given the role of web server by the system manager (http://www.ssuet.edu.pk/). Sometime thereafter, I assumed a faculty position at the International Islamic University Malaysia (IIUM). IIUM recently moved to a new 700-acre campus and has a modern fiber-optic 155Mbps ATM core network serving completely switched segments in various faculties. The responsibility of upgrading and organizing the computer labs at the Faculty of Engineering was given to me as the Computer Coordinator.

Figure 1. Faculty of Engineering Building

I set up two clusters of 70 new computers, all Dell PII machines served by two dual-CPU servers. In one lab, a Windows NT server was set up to provide Windows applications and printing services for students. In the other cluster, I installed and configured the server as a dual boot machine. By default, the

server boots into Linux, providing user authentication via NIS and user home directories via NFS. In case of Windows NT PDC (primary domain controller) failure, the NIS server can be booted as a backup PDC NT server.

Figure 2. Engineering Computer Lab

Server Setup

Since the Dell RAID controller is unsupported, we had to exchange the RAID controller for a tape backup unit so that Linux can recognize the hard disks. The **fdisk** print command now produces the output shown in Listing 1.

Listing 1.

In order to set up the NIS server, called garden, we had to overcome many minor problems. Using the search facilities at Dejanews (http://www.deja.com/), solutions were quickly obtained from discussions posted on Usenet. The problem of shadow passwords was resolved by using the **pwconv** command to disable shadowing. One handy method to check whether NIS is working correctly is to issue the command:

```
ypserv -d
```

This prints all the debugging information when an NIS client tries to authenticate a user account.

SMP was easily enabled by uncommenting the **SMP=3D1** line in the Makefile and recompiling the kernel. The 128MB RAM was recognized by Linux once I added an **append=3Dmem=3D128M** line to the /etc/lilo.conf file and executed /sbin/lilo. Any spaces in this line cause it to go unnoticed.

I exported /home via NFS for two purposes. One was to provide the NIS users with home directories, and the other reason was to enable them to view

important announcements. For this purpose, the following line was added to the profile of users:

```
cat /home/motd
```

I configured a separate PII machine, called jasmine, as an applications server. Jasmine provides applications via NFS, including VLSI layout tools, Netscape, Scilab, GNUplot, Ghostview and others. In order to be an efficient NFS server for 30 PII clients, jasmine does not take part in NIS. A third Linux server, a Pentium, was configured to serve as the faculty e-mail and web server (http://eng.iiu.edu.my/).

Client Setup

All 30 PII clients in the cluster are dual-boot-configured to reduce the annoyance of Windows loyalists. Since the lab manager was not interested in learning Linux, I had to give him a cookbook of actions to transform a machine booted with Linux into an NIS client station. What follows is the recipe for NIS client setup.

After setting up a basic Linux system with the proper NIC driver, copy the following files from the “model” client:

```
/etc/hosts  {To avoid setting up DNS service
     in localized cluster}
/etc/profile  {To set up paths and issue initial
     commands for an NIS user}
/etc/host.conf  {To set up the host lookup order
     with NIS}
/etc/defaultdomain {To set up NIS domain name}
/etc/rc.d/rc.inet2 {To start ypbind using
     /etc/defaultdomain as domain name}
/etc/fstab   {To import NFS directories into
     existing mount points}
```

Add a line containing “+” to /etc/passwd, /etc/group and /etc/shadow files. Reboot and log in as a “test” NIS user to verify functionality.

This cookbook worked perfectly, as the manager was able to fire up NIS clients without needing to know the setup details. Later, a booting message was added for the convenience of users, so they are informed of the dual boot-property of the clients. All that is necessary is adding a few lines to the /etc/lilo.conf file:

```
#start LILO global section
boot=/dev/sda
message=/boot/boot.msg
prompt
timeout=100
```

The boot.msg file contains the following text:

```
Type "linux" or "nt" to boot into operating
system of your choice.
```

To set up the X Window System, I turned to XSuSE (http://www.suse.de/XSuSE/XSuSE_E.html) as they have the largest set of drivers for various cards. Accepting the default choices for most of the questions, I was able to start X quickly by filling in the proper card type, monitor **hsync** and **vsync** values and video RAM size.

Instead of changing the path in the global profile each time an application is added, I included the path /usr/local/apps/cad/bin in the profile. In this directory, I use shell scripts or symbolic links to add new applications. If an application needs to be run from its home directory, a shell script with the application's name will run from this path. For example, the application **exchek** shell script contains the following lines:

```
cd /usr/local/apps/exchek
/usr/local/apps/exchek/exchek
```

An alternate approach is to use symbolic links. For example, a symbolic link for Netscape was created with the following command:

```
ln -s /usr/local/apps/netscape/netscape netscape
```

Linux Applications at IIUM

Linux came in handy for providing lab sessions in the senior-level course ECE4330 VLSI Design. This course would have gone without any true labs, as the commercial VLSI (very large scale integration) layout tools are too expensive for us. I set up the MAGIC VLSI layout editor under X and supplemented it with SPICE (simulations program with integrated circuits emphasis) for small layout simulation and verification. As there is no licensing problem, students could use these tools simultaneously on several workstations.

After coming across articles in the January 1998 *LJ* about PVM (parallel virtual machine), I set up the PVM web course server on one of the Linux machines. I divided the students into several groups and allocated two Linux machines to each group. Their target was to set up one machine as the master and the other as the slave in a PVM environment, then test various programs. Although the students could not succeed fully in running various programs, they enjoyed working on such important concepts with full control over the computers.

Current Projects

My current projects include setting up a filtering firewall using Red Hat Linux and binding an HP workstation cluster to the Linux NIS server for user account authentication. Thus, we will be able to stop some wasteful use of the Internet, as well as provide centralized accounts for all UNIX users.

Dr. **Dr Junaid Ahmed Zubairi** (junaid@iiu.edu.my) is an Associate Professor of Electrical and Computer Engineering at the International Islamic University Malaysia in Kuala Lumpur. Besides teaching and coordinating computer labs, he has an interest in exploring the natural beauty of Malaysia with his wife and two kids.

Archive Index Issue Table of Contents

Advanced search

Focus on Software

David Bandel

Issue #66, October 1999

durep, dog, tkfileman and more.

I for one am glad the brouhaha surrounding the Mindcraft/Microsoft challenge has died down. I know many folks were up in arms at first about the results, and I'll offer a few of my own observations. First, I'd like to note that few sites run anything like the systems used in the benchmark, i.e., any operating system on quad-Pentium 1GB+ RAM machines. Most of my sites run on anything from 486-33s with 16MB RAM to single Pentium 450s with up to 256MB RAM, so I rarely see a dual processor machine. Second, I'd like to thank MS for providing the kernel hackers the opportunity to use these systems so they could identify bottlenecks that show up only on this type of system. Finally, I don't have a watch that measures time in nanoseconds or milliseconds or even tenths of a second. Even if a file comes three seconds later from a Linux server than from an NT server (NetBEUI is their native protocol after all, so they should be better at it), at least I know I'll get it. With NT, I'm never sure until after it arrives—if it does. I hope none of our gentle readers are numbered among those who flamed Mindcraft. I promise no benchmark software is included with this month's selections.

durep:http://www.hibernaculum.demon.co.uk/

The **du** command can be used to find out how much disk space you are using, but it can be quite verbose. It often dumps out more information than anyone could use, and in no particular order. **durep** will show you the files on your disk, and by default, sort the information according to file size from largest to smallest. As a bonus, it displays this data graphically and can also create web pages. It requires Perl.

dog:www.nl.linux.org/~wsl/dog.html

Well, you knew it had to happen—someone who doesn't like **cat** brings you **dog**. Being the skeptic that I am, I had to check this one out. Since cat has been around a long time, it must be sufficient for most uses—it has enough options, anyway. However, dog does add a few new options, such as **-l** which allows you to choose specific lines by line number, and **-rot=** which allows you to rotate letters using any number you choose, not just 13. You can also display the file with a $ character to mark each newline in the file. It requires glibc.

tkfileman:www.mindspring.com/~joeja/programs.html

Sometimes it's the simple things in life that make it worthwhile. This is a simple thing as well. Its author says he's stopped development on it, but I see little I would change. The one nice thing about Tcl/Tk is that it's not fussy about the GUI underneath it. This little file manager will gzip, bzip2, gunzip, bunzip2, as well as tar and untar. And copy, move, rename, etc., are also included. What more could you need in a file manager? Well, how about gnorpm? I would have chosen xrpm, and that feature can probably be changed easily. This is just one of several nice packages the author has lying about on his web site (including a much improved version of the tknotepad highlighted in this column a few issues back). Requires Tcl/Tk (tested against v8.0.4) and a GUI. It will also require the actual files for those commands you wish to run (tar, gzip, bzip2, mv, gnorpm, etc.) in your $PATH.

geneweb:http://cristal.inria.fr/~ddr/GeneWeb/

When I was a child, I heard the statement “Every family's got a skeleton in the closet”. My family has plenty of skeletons which aren't even that discreet. Since I've started using this program, more keep appearing. If you are interested in genealogy, this web program is for you. Completely self-contained, it allows you to use either geneweb's built-in web server or your own web server and geneweb's CGI script. The author is French, and geneweb has support for thirteen languages easily chosen from the start page. Handy, when half your family speaks only English and the other half only Spanish. It requires ncurses, libm, glibc, ocaml and camlp.

portsentry:http://www.psionic.com/abacus/portsentry/

This particular piece of software is a very nice complement to your firewall (packet filter or proxy) software. The use of programs such as **nmap** with stealth mode by script kiddies and slow port scans, etc. make detection of probes difficult to spot. Older packages, such as **courtney**, aren't up to the task. **portsentry** doesn't just report scans—it logs them and reacts to them. If someone is probing ports where you aren't offering services, it will react to those addresses and drop their packets. The author has developed a well-

thought-out program and extensive documentation. This one is a must for the security-minded. It requires glibc.

DownLoader for X:http://www.krasu.ru/soft/chuchelo/

If you know what files you want, this is a very handy tool. It won't show you what's available, but if you point it at a directory, http or ftp address, it will attempt to download the files at that location. Complete with timeouts and retries, DownLoader attempts to optimize downloading large numbers of files. It would be nice to have a listing option and perhaps be able to choose particular files. Its difficulty of use is mitigated by its ability to snarf both ftp and http. It requires libpthread, libgtk, libgdk, libgmodule, libglib, libdl, libXext, libX11, libstdc++, libm and glibc.

gtkpool:http://members.xoom.com/jacquesft/

gtkpool is a pool game for the X Window system that allows you to play a game of billiards for relaxation. What it has been able to confirm for me is that my eye is as crooked as I have always thought—my bad shots are not really imperfections in the pool table (my favorite excuse). If you enjoy playing pool, this game will keep you amused when the pool hall is closed. It requires libgtk, libgdk, libgmodule, libglib, libdl, libXext, libX11, libstdc++, libm and glibc.

David A. Bandel (dbandel@ix.netcom.com) is a Linux/UNIX consultant currently living in the Republic of Panama. Co-author of *Que Special Edition: Using Caldera OpenLinux*. 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

Letters to the Editor

Various

Issue #66, October 1999

Readers sound off.

Portable MP3 Players

I read with interest the article in the June issue (#63), “MP3 Linux Players” by Craig Knudsen. It seemed to imply the Empeg product was the only solution for MP3 players in cars. I would like to suggest braver readers take a look at http://www.mp3car.com/. The site has great examples and links on how to build one yourself. Thanks.

—Alistair Hedge ahedge@skylink.net

Neural Networks

Great article! Thanks for printing “Stuttgart Neural Network Simulator” by Ed Petron, July 1999. I have been working with neural networks for two years now in my high school science fair projects. This article provided a good introduction that I would have loved to have had handy when I first started working with neural networks. I really enjoy seeing Linux being used not only as an alternative OS, but as an OS with scientific and educational purposes.

—Michael Katz-Hymanm katzhym@nhgs.tec.va.us

Free Beer

The first thing I did after reading the Guest Editorial entitled “The Point Really is Free Beer” was to check the cover date on the magazine to see if it was an April Fool's joke—sadly, the issue date was July 1999.

People like Eric Hughes lead the Open Source movement in the same way that the front bumper on my truck leads me down the highway: it's along for the

ride, but it really has nothing to do with who's driving the machine or how we get where we are going.

He states that “To be generous, maybe one-quarter of the total value of software comes from the product.” To see what a lie that statement is—it is only necessary to imagine his proposed institution *without* the software. What value does it have? The answer is *zero*. All of the well-dressed staff, administrators and planners are of no value whatsoever without the people who produce the product.

The converse is not true; programmers like Linus and the other open-source developers have great value to all of humanity without the participation of institutionalized parasites in the process.

Mr. Hughes points out that most of the work so far has gone into building software tools. Well, duh—first you build tools—then you use those tools to build applications. You can't do it any other way.

Mr. Hughes has the audacity to accuse those of us who write open-source code of having selfish ends. Wow! What about Mr. Hughes' goals? Assume that two-thirds of the 25 million dollar grant he wants to get the ball rolling would go to “the talent”. That leaves about 8.3 million. Building construction and furnishing will eat up most of that: can't look chintsy—have to look solid to impress the idiots.

That will leave about one million for staff salaries. Since I assume Mr. Hughes will be willing to *lead* us, I guess his take will be about five hundred thousand a year, with the rest to be split up among the other drones at the institution. Bah.

Instead of giving grants to useless institutions or to groups of programmers, why not give the whole thing to the individuals who do the work? I can promise you my needs are awfully small compared to some institutionalized thief. Fifteen hundred a month would keep me writing open source pretty much full-time. It is about time the worker bees realize they are the ones with the sting—not the drones.

One hundred per cent of the value of software comes from the product. Period—end of discussion.

—Bob Canup rcanup@hal-pc.org

Linux Resources

I was chatting with someone and mentioned that *Linux Journal* was just full of advertisers with affordable Alpha systems, and sent him to you. He couldn't

find a list of your advertisers. Sounds like a business opportunity to me. List hardware vendors and software vendors. Let me search for vendors by name and by product. Then I could choose to buy from your advertisers, or I could find one that sells what I am about to buy and patronize them.

Thanks for doing everything else right. The guy I was talking to will probably be subscribing now. I gave him four links to Alpha vendors on the Web right out of the handiest issue. (Then I quoted the rates, and he replied, “cheap”.)

—Duane Smecker telmer@ptw.com

A list of advertisers in each issue can be found on the web site Table of Contents page for each. This list includes links to the advertisers' web sites — Editor

Linux Install on Laptops

I was inspired by the informative article in July's issue by Daniel Graves, in which he described the installation of Linux on an IBM Thinkpad 750. Several months ago, I purchased a Sony VAIO 505 mainly driven by the form and weight factor. One of my first tasks with the machine was to install Linux on it. I have made the chronicle of the task available at www.seanet.com/~scout/linux505.htm for anyone who is interested. I appreciate your publication, as it provides a good balance of topic breadth and depth each month.

—Ted Ipsen scout@seanet.com

BTS

Concerning Red Hat 4.2/5.0/5.1/5.2 and Debian 2.0/2.1.... Both Red Hat and Debian used to support libg++-devel, which included some very useful C++ templates. Available documentation at www.debian.org says that libg++-devel will no longer be supported and that libstdc++-devel contains the functionality.

This isn't true! If you compare the contents of libg++-devel and libstdc++-devel, you will notice many files are missing. How can I build my C++ code on a new Red Hat 5.2 machine without having to rewrite all those useful libg++-devel templates myself?

For example, libg++-devel contains g++/String.h, g++/Random.h and g++/Regex.h, while libstdc++-devel does not. Red Hat used to ship with libg++-devel, but doesn't as of version 5.2.

—Steve Durst sdurst@110.net

Floppy Formatting

Occasionally, in “Best of Technical Support”, someone asks about formatting floppies from any directory except root, and the gurus respond that it can't be done. But the following changes work for me with Red Hat 5.1:

```
chmod 777 /dev
chmod o+w /dev/fd*
chmod 777 /usr/bin/fdformat
reboot
```

Since gurus can't be wrong, what am I missing?

—John C. Burgess burgess@wiliki.eng.hawaii.edu

Kudos!

I have been a Linux user since way back in 1997, and a subscriber to *LJ* for nearly as long. I just wanted to let you know how much I enjoy your magazine. I learn something useful in every issue—from “Take Command” to “At the Forge” and all the others. Despite what I have read in other letters, I think you are doing a great job of balancing the advanced, technical subjects with those of the novice, intro type.

Last month (#63), I was surprised and delighted to see Linux applied to my old career of archaeology, and this month (#64) I was surprised and delighted to see the spotlight on Linux in my current career in the graphic arts industry! Now, if you can find someone who has successfully integrated Linux and beer-making to write a short article....

Keep the great stuff coming!

—Mike Edwards medwards@ega.com

Standards, Linux Wars, etc.

Troy Davidson (August *LJ* letters) has been watching too many movies. His Highlands war exists only in his imagination. I've been using Linux for an even shorter time than Troy has, but have quite a different take on MS vs. Linux. Comparing Linux with Windows 9x is like comparing seagulls with the penguin: they are two different birds with different purposes.

Linux is a UNIX clone, and UNIX is a multi-user system. It has important work to do and it is probably not going to expend a lot of effort looking after *you*. It has many of “you” to look after; so probably, it won't treat you like your nanny, which is precisely what MS Windows sets out to do. Linux in its present form is not for the casual user; it is for the person who wants to master the system in

the way that only the availability of source code can provide, and there are many such people out here.

Nobody remains a beginner. As you gain experience, the user-friendly “features” of MS Windows become obstacles. Then you look for a better place to work. It's waiting for you—it's called Linux.

Linux is not ready to take over the desktop, but the server war is over and Windows NT is not the winner. (We have two local ISPs; one on NT, one on Linux. So I have personal experience regarding which one holds up better.) There is no mystery as to why the bedrock under the Internet is UNIX, Linux and FreeBSD. They got there first and offer the most. NT has a following in business intranets—the suits tend to stick together.

Richard Stallman's world of free software is going to prevail, and for a very simple reason: free access to source is going to create an abundance of local experts. When the casual user discovers that one local expert is worth 1000 e-mails to MS support, then Linux will begin to take over the desktop. MS will always have a market; someone has to look after the beginners.

Let us not call for “standards”. We don't need standards. We need more of the same creative anarchy that has got us to where we are. Spend more time reading code, Troy, and less time watching movies.

—Jack Dennon jdennon@seasurf.com

Helius Review

It was fun to read the review for the Helius DirectPC router. However, you could have tested the download speed easily. I had the same problem with my ADSL connection—frequently, the Web at large would be the bottleneck. To get a better idea of what your real pipe to the Internet is, set up multiple downloads from multiple sites. Sum the throughput of the varied connections, and you'll get an idea of what your incoming connection can support.

—Michael Rasmussen mikeraz@barley.patch.com

Archive Index Issue Table of Contents

Advanced search

DiskOnChip2000 and GPL.

In your interview with Hans Knobloch of IGEL in *Linux Journal* issue #64, Mr. Knobloch correctly states that “IGEL developed the Linux boot driver to boot Linux from an M-System's DiskOnChip Flash Memory.”

However, I was extremely surprised to read in the next sentence that “The driver has been released under the GPL to the Linux community and can be downloaded from M-System's server”.

This is *not* the case. The kernel driver for the DiskOnChip 2000 is a binary-only driver. Any person or company distributing a kernel with the DoC driver linked into it is in violation of the GPL and s leaving himself open to legal action for a violation of the licensing terms of the Linux kernel.

We were looking seriously at the DiskOnChip for our own embedded solutions, but we decided we were not able to use the driver provided by M-Systems, because of this and other technical reasons.

The current advice to the Linux community is to avoid this device at all costs. More information on this subject is available at http://www.linux-mtd.infradead.org/doc2000.html. Thanks for your time.

—David Woodhouse, David.Woodhouse@mvhi.com

To avoid further discussions about IGEL's part of the DoC boot driver (a part which does nothing if used alone and is only a piece of the M-Systems complete boot driver), we at Infomatec IGEL Labs decided to rewrite this part and convert it into a loadable module instead. This should end the discussions of what is the right GPL license term for this piece of software.

I hope this focuses the discussions on the tremendous potential of the IGEL Flash Linux Technology (JNT) as an embedded Linux OS, rather than getting excited over a piece of software that we supplied to M-Systems to accommodate their product.

Also, I'd like to correct the misprinted WWW addresses of our company. The correct listings are:

Americas and Canada: http://www.igelusa.com/
Infomatec IGEL Labs GmbH: http://www.igel.de/
Infomatec AG: http://www.infomatec.com/

Thank you very much for your continued support. Keep up the good work!

—Hans L. Knobloch, President & CEO, IGEL LLC

Re: DiskOnChip2000 and GPL

Please note that my advice regarding the DiskOnChip has now changed. M-Systems have given me complete specifications for both the device and the flash filing system they use on it, along with sample hardware.

I now have a real driver available under the GPL—it's still alpha-test quality, but it's getting there quite fast.

See http://www.linux-mtd.infradead.org/doc2000.html for more details.

—David Woodhouse, David.Woodhouse@mvhi.com

Comments on June 1999 Article "Root File System on RAID"

I enjoy *Linux Journal* and have been a regular subscriber for over two years. I thank *Linux Journal* for the June 1999 Article "Root File System on RAID." I like to see columns of this type and hope to see more like it in the future.

However, this article is now a completely out of date representation of the software RAID subsystem in Linux. I'm wondering how long this article sat around in *LJ* offices before it was published.

To point out a few wrong things about it:

- LILO and Loadlin can now boot *directly* off of a RAID0 or a Linear device.
- The syntax and commands used are almost completely wrong in this article for the modern RAID subsystem. The syntax he uses refers to a much older release of the raidtools package.
- The kernel can now autodetect RAID-0,1,4,5 devices and activate them at boot time, making them possible to mount as a root file system without much hassle.
- The whole business that he describes to make a root RAID file system is largely irrelevant thanks to the above point.

These are not changes implemented in the last month or two, many of the features he missed have been part of the RAID subsystem for 6 months or more. And to show you the age of this article, the "Resources" section makes references to software and patches now almost 2 years old, both the raidtools package and the kernel patch mentioned were released in October 1997.
A modern RAID HOWTO can be found at http://ostenfeld.dk/~jakob/Software-RAID.HOWTO/
Modern RAID tools and patches for kernels can be found at ftp://ftp.us.kernel.org/pub/linux/daemons/raid/alpha/

I thank *Linux Journal* for the article and know how hard it is to write on a fast-moving topic like this, but please make sure of its timeliness when it is published. Thanks.

—Michael Wise, wiseman@infinet.com

> *Actually, this article had a very short shelf life at our office—two months from when it was received to when it was printed. Sorry that it turned out to be out of date. Thank you for sending us this additional information. —Editor*

More on PCI Modems—LJ August 1999—#64

Greg Bailey wrote, that he managed to get a PCI modem to work without issues. Recently, “The Computer Paper” (August 1999 Volume 12 N.o 8) on page 60 had a clipping *Linux support for all modems*.

A company called Actiontec Electronics claims to support Linux across all its modems: ISA, PCI and PC Card.

Actiontec is at http://www.actiontec.com/ and “The Computer Paper” (A free Canadian Publication is at http://www.tcp.ca/.)

I do not know this company, or its modems, but it seems that we can no longer groan at the mention of a PCI modem. Apparently some will work.

—Tim Legge, tlegge@fundy.net

“Compounding Errors?” (letter, 8/1999)

I was disappointed but not surprised by your reply to the letter from Greg Bailey in the Letters section, *Linux Journal*, August 1999, regarding PCI modems under Linux.

I have found this topic to be a very confusing one, although I must admit I was taken aback by the reply that “everyone I talked to” agreed PCI modems were incompatible with Linux. Perhaps you need to widen your circle of acquaintances? :-)

In any event, there is a dead-bang-giveaway that you can check for—on the outside of the box, before you buy. Look to see which operating systems the modem supports. If it supports either Windows 3.1 or DOS (or both), it supports Linux. If it doesn't support either of these operating systems, it almost certainly will not support Linux, as in that case it almost certainly is a dreaded Winmodem.

Take good care,

—John Freed, john@integrita.com

Red Hat 6

Am I the only one who finds this latest copy of Red Hat (6 something) less than ideal? Two programs I am fond of, Midnight Commander and locate, are provided in advanced broken versions, and I've managed to screw-up the file system many times without trying very hard, something I *never* did with 5.2.

Linux ignoramus I may be, but I think this product is getting *too* much like Windows.

Best wishes

—James G. Owen, 71121.625@compuserve.com

Review of Caldera 2.2 and Slackware 4.0

Having been very impressed with Linux for the past several years, when the latest releases came out I decided to upgrade my various systems. My prime computer for development work is a AMD-K6/200 with 32 MB RAM and 2 GB hard drive dedicated to Linux. Swap space is a 120 MB drive. I had installed OpenLinux 1.3 on this system with no trouble. OpenLinux 2.2 installed acceptably, but I could not get the printer or LAN connection to work. Finally I got the printer going by ignoring the COAS and LISA routines and manually configuring the Epson filter for Ghostscript, etc., with MagicFilter.

I could not get OpenLinux 2.2 to properly install on a Pentium P-90 with 16 MB, nor on a 486/DX-4/100 with 16 MB and a small hard drive. Indeed, even using the LISA install according to the manual was difficult and troublesome. Yes, I know that Caldera calls out for 32 MB RAM, and this restriction is indeed necessary.

In particular the “cutsy-poo” graphics distract from the job of installing. I found little control over the install using the Lizard. Word Perfect printer drivers, etc., were not properly installed, nor even prompted. Thus, I would be hard pressed to recommend OpenLinux 2.2 for someone who is hardware impaired or who uses an Epson dot-matrix printer, or even an older HP LaserJet. Yes, the available printer drivers for the system are limited to the more recent printers.

Caldera does not give an upgrade path from OpenLinux 1.3 to 2.2. Fortunately the Zip drive works well, so backing up the necessary files was not a problem.

On the positive side, the X Window System configured easily, and KDE installed with quite acceptable defaults. The kernel seems faster, allowing more efficient processing. Getting the Zip drive to work after the installation was complete was as simple as loading the driver.

On the other hand, I also had been using Slackware 3.6 on a system at church (I am a pastor). I decided to upgrade that system (a 486/DX-100

with 16 MB and 1.6 GB dedicated to Linux) with Slackware 4.0. The installation was not as graphic or as entertaining, but it worked beautifully with very little tweaking. I was able to upgrade rather than do a complete new install, saving lots of time and frustration. (Yes, the backups were done just in case.)

Of the two products, I would recommend Slackware. Once running, both seem to be fine systems, but the installation of Slackware was by far easier and more controlled.

—Jeff Williams, cfiaime@mpks.net

My Linux Journal Subscription

When I arrived home from work today, I found the latest issue of *LJ* waiting for me. As usual, the back page was nearly torn off, the binding was broken in a few places, and the cover had several creases in it. I have been a reader of *LJ* since issue 5 or 6 and a subscriber since issue 12. I look forward to reading the issues whenever they arrive, but I am getting *very* tired of paying for a magazine that comes to me in this sorry state. Of all the subscriptions I receive, your magazine is the only one that consistently arrives damaged.

You need to do something about the packaging/shipping of your magazine or you are likely to lose my business. I assume I am not the only subscriber who has these problems.

—Richard A Spaller, rick@amercyb.com

> *We've done several things in the past to try to help with this problem. We now use much thicker paper for our covers, and we have changed printers and distribution centers. I'm sorry that you still have a problem. We'll keep trying. — Editor*

Minor Correction

At the end of Marjorie Richardson's interview with Red Hat CEO Robert Young in the August 1999 *Linux Journal*, you stated that Red Hat was going public and gave an address for a web page containing Robert Young's press announcement. The address you gave - http://www.redhat.com/corp/press_iop.html - is wrong. Probably a typo, but the correct address is http://www.redhat.com/corp/press_ipo.html (note: ipo, not iop). It's a minor error, but I had to go through Red Hat's web site to find it, so I thought I'd point it out.

—Brad Mitchell, bcmitch@learnlink.emory.edu

Black Bars and Slick Paper

In the August issue 'Letters' column, a reader complained about the black bars you put around certain sections; he didn't like the ink getting on his

fingers, or some such. You replied about the good reason you put those bars there, and that others also had concerns, and that you were working with the printer to discover a better solution.

Having worked on more than one UserGroup news magazine, I'd guess that the real problem is the slick shiny paper. While it makes for crisper colors and graphics, it also tends to resist the inks, allowing them to smudge all over the place. I've noticed it on humid days that wherever I've had my thumb, the print becomes smeared and difficult to read. I also find that in most lighting conditions, the slick paper generates a glare that makes reading difficult and uncomfortable. I'd vote in favor of keeping the black bars and moving to a flat finish paper. Being that this is a computerized world, a URL pointing to where I can compare screen shots on my own system has more value than crisply printed versions; I'm more interested in the text of the articles than the trueness of the colors.

Thanks for being a wonderful collection of data and information. BTW, I just got your Sept. issue, and wanted to share the comparison chart with a friend, but that chart seems to be missing from the online edition.

—Bruce Kingsland, brucek@pacifier.com

> *The chart is there now. We are still looking for solutions to the black ink problem. —Editor*

You really need to sort out the black ink problem

I seem to recall seeing a letter from one of your readers in either the last issue, or the issue before, complaining about the black ink that your printer uses to produce the magazine. The text generally seems to be OK, but the black adverts are terrible. I too am getting very tired of leaving black fingerprints all over the inside of my magazine, and my mouse, and my keyboard...

This month's big culprit - the Yellow Dog advert on page 107, next to the letters page, funnily enough.

I like the magazine enough to subscribe to it, and I see it on the shelves often enough to know that there are a lot of people out there buying it in sufficient numbers for the book shops to keep stocking it. If the printer you have can't rectify this problem, and you can't ban black ads because of the revenue they bring in, then find a different printer. Bye for now,

—Stuart, spowell3@mmm.com

Reader's Choice Quote

I just received the September issue of *LJ* in the mail. In the sidebar on the Readers' Choice Awards you say: “In the immortal words of James Hoffa 'Vote early and vote often'.”

FYI this has been a saying in Chicago politics since the 1960s when the late Mayor Richard J. Daley (The Boss) was in power. (It may even pre-date that going back to the 1920's and Mayor Big Bill Thompson).

—Bill Tate, liverpool@mindspring.com

> *Sorry, I always understood it to be Hoffa, but can't remember now where I found it so admittedly I could be wrong. Thanks for the correction. —Editor*

Archive Index Issue Table of Contents

Advanced search

Is KDE the Answer?

Phil Hughes

Issue #66, October 1999

I see two problems: is looking like MS Windows good or bad, and is KDE a better answer than GNOME? Let's look at these one at a time.

Well, it depends on the question. Douglas Adams fans know the answer is 42, but they also know it was much harder to find the question.

KDE is largely the same. It offers a nice, user-friendly GUI. It can even look enough like MS Windows so that converts don't know they are being converted.

I see two problems: is looking like MS Windows good or bad, and is KDE a better answer than GNOME? Let's look at these one at a time.

Initially, I thought looking like MS Windows was good. After all, by looking like something people are used to, we get converts. In practice, however, I don't think this works. Sure, if you know how to click on “start browser” in the Windows environment, you know how to click on the same function in KDE. Unfortunately, it seems as if the actual result is that you are encouraging what I would call stupidity, but most would call computer illiteracy.

Why so? Because you have taken a group of people who think clicking on “start browser” means they understand computing and the Internet and reinforced their conclusion. It works on Windows and it works on Linux, so it must be true. Right? Nope, very wrong. All we have done is say, “If you don't get it, you can still use Linux.”

Don't get me wrong; this is a perfectly valid group of users. Building an “Internet appliance” based on Linux to appeal to this group seems like a good thing. After all, they need it, and Linux offers more functionality at less cost than a Microsoft-based alternative. However, this doesn't mean these people are now computer literate.

This is similar to the stories being told in the early '70s about how idiot-proof databases and such would eliminate the need for programmers, as I remember, by 1976. What happened? Did we not make all the software we needed? Programmers are still in demand, if for no other reason than to continue to write more idiot-proof programs as we watch the development of more sophisticated idiots.

Encouraging people who aren't computer literate to think they are is as dangerous as encouraging people who don't know how to drive to think they do. In either case, we end up with clueless people out there driving, whether on the streets of Ballard or on the Internet.

The next question is, is there anything we can do? Is there a way to make the streets or the Internet safe again? All we have to do is build that Linux-based appliance I talked about. Tell the users they are using a crutch—that they don't have a clue, but we are making up for it, using a Linux-based appliance. They will thank us.

For the rest of us—people who do need the power and grace of a real operating system—Linux is the right answer again. The difference is we know that underneath the “start browser” button, something is happening. We understand that armed with a terminal window and some knowledge of awk, grep, Perl, Bash, gcc, Python and much more, we can do some real computing work—like writing a better “start browser” button.

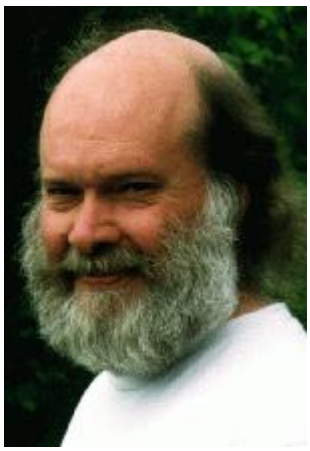
Now, on to the second question: Is GNOME a better answer than KDE? This is like asking if Apple picked the right kernel with NetBSD vs. OpenBSD, FreeBSD or Linux. It doesn't matter. They are all reasonable choices, and it is better to get on with life than go back and review already-made decisions.

In the KDE vs. GNOME war, KDE is ahead, offering more functionality and more programs than GNOME with development continuing at an amazing pace. The glitch in KDE development was the non-open licensing of Qt. In fact, that is what created GNOME development in the first place. With Qt 2.0, it is a non-issue. The Qt people saw the problem and created a new license that addresses it. To me, it seems hard to complain that the Qt developers think if you sell your product, you should have to pay for Qt. While it isn't the GPL, the idea of “if you are free, we are free” seems a fairly good rule to live by.

Most distributions have adopted KDE as their default and, as far as I know, all distributions include KDE. It isn't perfect, but is very usable. The kinds of problems you encounter are things like the default paper size being **A4** and it taking a recompile to change it to **letter**. These kinds of bugs can be crushed quickly if we just get behind it and move it along.

Okay, I have put on my flame-proof suit, so I'm ready to say it. If we all, including the GNOME people, jump on the KDE bandwagon, we can soon create a better product with more features and fewer bugs. And, remember those appliance users I talked about? If we offer one standard GUI rather than two, it will be much easier to show we have something that might interest them.

Should the GNOME people be offended? No, not at all. Much as the GNU project helped make Linux happen, the existence of the GNOME project forced the rethinking of the Qt license and has contributed valuable ideas to KDE. By working together, everyone, including the application developer and the appliance user, benefits.



Archive Index Issue Table of Contents

Advanced search

Best of Technical Support

Various

Issue #66, October 1999

Our experts answer your technical questions.

Modem Question

Can I get two nearby modems to talk over a short connecting line without going through the telco, i.e., going from the phone line connector of one modem to the other with a twisted pair (reversed)? It seems this should work and I think I know the correct AT commands, but I can't get it to work and nobody seems to know the answer. I am not talking about a null modem cable. I want to use this to test out PPP server setups. Thanks for any information. —William Strickfaden, xws99@hotmail.com

This depends on your modems. The phone company normally provides line voltage that modems modulate with their signals to talk to one another. Some modems have been known to work without this voltage, but before you go that route, you can look for a “phone-line simulator”. This device will provide the line voltage and will even ring one end when the other picks up. They're usually not too expensive, or if you're a hobbyist, you can try to build one. Many circuit encyclopedias have simple circuits which do that.

This is the only option. LAN signals have voltage supplied by the devices on each side. You cannot do the same thing with phone connections just by reversing the wires. —Chad Robinson, Chad.Robinson@brt.com

Secure Passwords and PPP Connection

Has anyone had success in establishing a reliable PPP connection from Linux to a Shiva LanRover modem bank(www.shiva.com/remote/d56)? I've been through DejaNews and talked to the Red Hat PPP guru, but so far no luck.

Also, my employer uses SecurID to form part of the password, so it is literally impossible for me to have a static password entry in /etc/ppp/pap-secrets or /

etc/ppp/chap-secrets. Is there any way to accommodate passwords that are generated through SecurID? —Steve Masticola, masticol@scr.siemens.com

The **dip** program has scripting features that include explicit support for SecurID cards. It even understands about the static and variable parts of SecurID passwords, with support for both pieces independently. The script simply says:

```
securidf fixedpart
```

to store the fixed part. Later, the script says

```
securid
```

whenever the password prompt is recognized; this makes the script stop and prompt you for the variable part, and then dip sends the combined parts to the server. —Scott Maxwell, s-max@pacbell.net

Try using **seyon** or **minicom** to see if you can get a login and password prompt. Type **ATDTnumbertocall** then press **enter** once you get the CONNECT message. If you can, you should use those to connect, close the application without hanging up the modem, then run **pppd** by hand:

```
pppd /dev/modem 57600 crtscts modem noipdefault\
defaultroute
```

If you don't have a /dev/modem link, substitute it for your serial port. —Marc Merlin, merlin@varesearch.com

LILO Problems

I am a new user. I installed Red Hat version 5.01 for the first time on a partitioned disk (486 system). Immediately after I switch on the message, “Lil” comes up and the system stops. When I use a floppy boot, it is all right. Should I reinstall Linux again, or can I configure properly after it is booted through the floppy? The other partition is Windows 95. I want both options to be displayed while booting. Please help. —Manilal, chirakkal@yahoo.com

You must have had a problem with configuring LILO during the Red Hat install. One possible reason is that your root partition crosses the 1024 cylinder boundary on your disk, and this prevents LILO from booting. There are many other possible problems with IDE disks and the different kinds of translations that your BIOS may be able to do (CHS or LBA head translation). I suggest you try different settings in the BIOS and the linear option for LILO. Also refer to the following LILO HOWTOs:

```
metalab.unc.edu/LDP/HOWTO/mini/LILO.html
metalab.unc.edu/LDP/HOWTO/mini/Multiboot-with-LILO.html
```

—Marc Merlin, merlin@varesearch.com

Slow Transfers

I need to mount a 6GB HDD IDE drive formatted for NTFS on my Linux system that will allow me to transfer a 5GB file from the drive to Linux format, so that I can process the data in the UNIX world. I have found a read-only NTFS drive that is under development, but I could not get it to compile. I get an error on a **time_t** call. I've tried to use FTP to transfer the file, but it goes so slow I'm afraid it would take a week to do the transfer. I also tried to FTP the file to my RS/6000, but that consistently dies at 1.2GB. If I try to use the SAMBA interface, the transfer dies at about 500Mb.

I've also installed AWK on the NT box, but it goes so slow that I can't believe the NT box will stay up long enough to complete the transfer. Help! —Algis Posius, algis@pacbell.net

The 2.2.x kernels already have support for both read/write in NTFS partitions. Even though I've never tried to copy that amount of data, it seems stable enough for the task. Just one question: do you have 5GB of data or a 5GB file? If you have a 5GB file, you are out of luck. The 32-bit version of Linux (and AIX also) can address files only up to 2GB. —Mario Bittencourt, mneto@argo.com.br

Choosing the Right Commands

I'm trying to configure my Red Hat 6.0 system to allow clients to access CD-ROM images from my Linux server hard drives. After looking at various file systems such as Samba and NFS and commands such as MAKEDEV, vnconfig, mount, smbmount etc., I'm getting confused as to which combinations of commands to use. —Mark J. Foucht, mark.foucht@eds.com

Red Hat 6.0 uses **knfsd**, which works somewhat differently compared to the old userland NFS server. One big difference is that you have to export each file system mounted in order for clients to see them (with the old server, you could just **export= /**, and clients would have a view on all your file systems).

In your case, if your CD-ROM is mounted under /mnt/cdrom, put the following in your /etc/exports file:

```
/mnt/cdrom (ro)
```

Then, type the following to migrate the entry to /var/lib/nfs:

```
/xtab
moremagic:~# exportfs -av
exporting :/mnt/cdrom
```

To see if it worked, type:

```
moremagic:~# showmount -e localhost
Export list for localhost:
/mnt/cdrom (everyone)
```

To mount from another machine, type:

```
mkdir /mnt/remotecd
mount remotemachinename:/mnt/cdrom /mnt/remotecd
```

—Marc Merlin, merlin@varesearch.com

If your CD-ROM will be used by Windows computers, you should use Samba. Here is the entry you can add to your /etc/smb.conf file:

```
[CDROM]
comment = CDROM
path = /mnt/cdrom
read only = yes
guest ok = yes
case sensitive = no
mangle case = yes
preserve case = yes
```

You should restart Samba after modifying the file. Just type as root:

```
/etc/rc.d/init.d/smb restart
```

If you want to make it accessible to NFS users (UNIX computers), you should add the line **/mnt/cdrom** to your /etc/exports and restart your NFS daemon by using **/etc/rc.d/init.d/nfs restart**. —Pierre Ficheux, pficheux@com1.fr

Memory Error, Parallel Computing

I am currently running Caldera Openlinux 1.3 on a Compaq Presarion CDS 526 (486 66MHz). Believe it or not, I had no trouble getting it loaded on my machine. I do have a problem with my RAM memory. When I do the **free** command, it shows I have only 15MB of memory, when I actually have 36MB. Why is this? Is it a problem that has been corrected in a more current kernel, or is it more of a hardware problem?

My next question concerns the world of parallel computing. I have a new computer on order (P3 500MHz), and when I get it, I will be installing Linux on it as well as the one mentioned above. I am interested in hobbying in the world of parallel computing, and I wondered if it would do any good trying to run parallel with a 500MHz machine and a 66MHz machine, or will the whole thing run slower? Thanks for your help. —John, johnwtaylor2@hotmail.com

This 36MB you've mentioned is a rather “non-standard” amount of memory. Please use a **dmesg** command to see how much memory it finds during boot time. —Mario Bittencourt, mneto@argo.com.br

There is a kernel option for limiting the memory to 16MB; maybe it is activated in your current kernel. You should recompile a new kernel without this option, in “General Setup”: Limit memory to low 16MB (CONFIG_MAX_16M) [N/y/?] N —Pierre Ficheux, pficheux@com1.fr

On your second question, it will depend on how you do it. Think of a job jar, representing a problem decomposed into independent jobs. Each CPU grabs a job out of the jar when it's finished with the previous job. With good choices of job sizes, you win. If the jobs are too small, the extra communication and coordination overhead negates the gains from the slower CPU. If too large, the faster CPU will finish first and have to wait for the slower one to finish—and it may end up waiting longer than if it had done all the work itself. You may have to experiment to find good job sizes, though the obvious computation based on the two systems' relative speeds should get you in the right neighborhood.

I'd recommend you start by looking into PVM, the Parallel Virtual Machine system. Find PVM at www.epm.ornl.gov/pvm/pvm_home.html. —Scott Maxwell, s-max@pacbell.net

Correction

Archive Index Issue Table of Contents

Advanced search

New Products

Ellen M. Dahl

Issue #66, October 1999

AltaCluster Systems, EasyCopy 6.0, ArahWeave 3.0 and more.

AltaCluster Systems

Alta Technology announced their family of AltaCluster systems. Using Pentium III processors at speeds of up to 550MHz, the AltaCluster combines high performance and unique packaging to form scalable and reliable multi-processor systems. The eight standard AltaCluster compute nodes come complete with up to 1024MB of synchronous DRAM, a 6.4GB UltraDMA EIDE hard disk drive and 10/100 Ethernet connectivity. Systems are stackable and scalable from one to thousands of processors. AltaCluster systems are fully integrated with Linux and implement Parallel Virtual Machine and Message Passing Interface technology. See the web site for pricing options.

Contact: Alta Technology, 8689 S. 700 West, Sandy, UT 84070, 801-562-1010, 801-568-1010 (fax), sales@altatech.com, http://www.altatech.com/.

EasyCopy 6.0

AutoGraph International announced EasyCopy 6.0, a major rewrite of AGI's flagship product EasyCopy/X. The software provides integrated and flexible tools for working with image data and printing. EasyCopy 6.0 includes a new GUI, an integrated image viewer and a fast image browser. It offers complete control of the printer and improved selection of page layout and color options, imports a wide range of common CAD and DTP image file formats and has more tools for system administration. Free evaluation copies can be installed from the CD-ROM. EasyCopy pricing begins at $395 US. The AGI suite of image communication tools is available for all UNIX platforms including Linux, on one CD-ROM.

Contact: AutoGraph International Inc., 1782 Technology Drive, San Jose, CA 95110-1306, 408-436-7227, 408-436-7255 (fax), sales@augrin.com, www.augrin.com.

ArahWeave 3.0

ArahWeave is an integrated dobby and jacquard CAD/CAM system for weaving designs. The software supports Linux and is useful for design studios, art schools and weaving mills which use dobby or jacquard weaving technologies. With looms and CAD connected to a network, designers can make realistic fabric simulations by entering technical data. Please contact Arahne for fabric samples or pricing.

Contact: Arahne d.o.o., Staniceva 17, SI-1000 Ljubljana, Slovenia, +386-61-1395-280, +386-61-1316-119 (fax), arahne@arahne.si, http://www.arahne.si/.

CAD-UL C/C++ Toolkit for Linux

CAD-UL, Inc. announced an embedded-centric C/C++ compiler and toolkit for Linux developers. Engineers may prototype their code on a Linux-based host platform using C/C++ tools designed specifically for embedded development. The CAD-UL C/C++ Toolkit for Linux features support for protected-mode programming and real-mode programming and is available at a base price of $5,600 US.

Contact: CAD-UL Computer Aided Design Ulm GmbH, Lämmerweg 32, D-89079 Ulm-Einsingen, Germany, 480-945-8188, 480-945-8177 (fax), sales@cadul.com, http://www.cadul.com/.

CoffeeCup HTML Editor++

CoffeeCup Software announced a Linux version of its full-featured GTK Editor. CoffeeCup includes many useful features such as 60 JavaScripts, Quick Start Wizard, Table and Frame designers, a huge HTML tag vocabulary, 30 background images, more than 175 animated GIFs, about 140 web-icon graphics, a snippet editor, project directories, a horizontal rule designer and a link wizard. Priced at $40 US, the HTML Editor++ is compiled against shared libraries as well, so if the GTK 1.2.x version is installed, running the software is as trouble-free as making a pot of coffee.

Contact: CoffeeCup Software, Inc., 801 Elizabeth Street, Corpus Christi, TX 78404, 361-887-7778, 361-887-8788 (fax), sales@coffeecup.com, http://www.coffeecup.com/.

Funnel Web 3.5

Active Concepts announced a Linux version and an upgrade of its web-site analysis solution, Funnel Web. The web profiling tool is designed to assist businesses and organizations in tracking vital information on customer usage patterns, market penetration and effectiveness. It features Streaming Analysis, domain aggregation, offsite processing, event messaging, Proxy analysis, regionalized reporting and a choice of command-line interface or GUI. The Standard version of Funnel Web for Linux is priced at $249 US, the Professional version at $499 US. Upgrades and crossgrades range from free to $299 US.

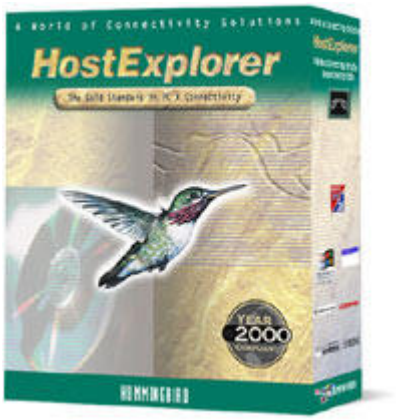
Contact: Active Concepts, 159 Pelham St., Carlton, Victoria 3053, Australia, +61-3-9348-2122, +61-3-9347-9914 (fax), info@activeconcepts.com, http://www.activeconcepts.com/

GoAhead WebServer 2.0

GoAhead Software released GoAhead WebServer 2.0, an open-source, royalty-free, standards-based Web server designed specifically for embedded systems. GoAhead WebServer 2.0 uses Active Server Pages (ASP), embedded JavaScript and in-memory CGI processing to deliver dynamic web-page creation. New features of version 2.0 include ports to more operating systems and an improved user interface. GoAhead WebServer supports Linux and many other platforms. Licensing information and source code are currently available for free download from GoAhead Software's web site.

Contact: GoAhead Software, 10900 NE 8th Street, Suite 750, Bellevue, WA 98004, 425-453-1900, 425-637-1117 (fax), info@goahead.com, http://www.goahead.com/.

HostExplorer Web and Exceed Web



Hummingbird Communications Ltd. announced version 2.0 of HostExplorer Web and Exceed Web, its Web-to-Host and Thin X solutions which give expanded access across the Internet, intranets and extranets. HostExplorer Web, Exceed Web and JuMP, the Java-based middle-tier management platform, are vital connectivity technologies which allow easy access to IBM mainframe, AS/400, Linux and UNIX hosts by sharing business-critical information with partners, employees and mobile users across extranets. HostExplorer Web pricing starts at $127 US per user, for a 50-seat license. Exceed Web pricing starts at $296 US per user for a 10-seat license.

Contact: Hummingbird Communications Ltd., 1 Sparks Avenue, North York, ON M2H 2W1, Canada, 877-359 4866 (toll-free), 416-496-2207 (fax), sales@hcl.com, http://www.hummingbird.com/.

Token-Protected VPN Server

CRYPTOCard and InfoExpress announced the release and complete integration of CRYPTOAdmin 4.0 and VTCP/Secure 4.0. The joint security software solution creates a simple, secure and affordable extranet-friendly Virtual Private Network with embedded and automated strong authentication. The CRYPTOCard-InfoExpress token-protected VPN server and the client software run on Red Hat Linux. There is no annual license or maintenance fee for CRYPTOAdmin 4.0 or CRYPTOCard tokens, which do not expire. Pricing for InfoExpress VPN client software starts at $89 US per seat and the Gateway servers range from $1,495-$2,495 US. CRYPTOCard's RB-1 has a suggested retail price of $79 US, software tokens $59 US. Pricing for CRYPTOAdmin 4.0 Standard Edition is $5,000 US and includes the easyRADIUS 4.0 authentication server.

Contact: CRYPTOCard, Suite 304, 300 March Road, Ottawa, ON K2K 2E2, Canada, 613-599-2441, 613-599-2442 (fax), info@cryptocard.com, http://www.cryptocard.com/.

Contact: InfoExpress, 425 First Street, Suite E, Los Altos, CA 94022. 650-947-7880, 650-947-7888 (fax), info@infoexpress.com/, http://www.infoexpress.com/.

Magic 8.20 for Linux

Magic Software announced a new version of its application development tool. Using Magic, one can quickly develop enterprise-level applications for deployment on Linux. Applications created with other technologies can also be ported to Linux with little effort. For a limited time, Magic Software will offer free limited development and deployment licenses of Magic for Linux. The package includes a Magic/Linux Development Kit with a Magic v8.20 limited license and Magic for Linux, limited e-mail support, and full documentation. The version of Magic being used for Linux is the standard version of Magic 8.20, which supports both Informix and Oracle databases. It has been tested with Red Hat versions 5.1 and 5.2.

Contact: Magic Software Enterprises, 5 HaPlada St., Or Yehuda 60218, Israel, +972-3-5389376, +972-3538-9333 (fax), linux@magic-sw.com, http://www.magic-sw.com/.

NAGWare f95 Compiler

The Numerical Algorithms Group (NAG) issued release 4.0 of their NAGWare f95 Compiler. NAG compilers are available on a wide range of platforms including Linux. The NAGWare f95 Compiler features extensive compile time and runtime checking, Allocatable Components and IEEE Floating Point Exception Handling. Price varies depending on your hardware; contact the sales department.

Contact: NAG, Inc., 1400 Opus Place, Suite 200, Downers Grove, IL 60515-5702, 630-971-2337, 630-971-2706 (fax), naginfo@nag.com, http://www.nag.com/.

P1000 Gigabit Ethernet NIC

Phobos Corporation announced shipment of its P1000 Gigabit Ethernet Network Interface Card (NIC) with support for PhobosLink Trunking Software on Linux/UNIX and NT servers. The P1000 is a 32/64-bit PCI NIC that connects to multimode fiber optic network cable systems through an SC fiber interconnect. PhobosLink Trunking Software, included with the card, allows network managers to aggregate the bandwidth of up to four P1000 NICs to a maximum of four gigabits per second. Network traffic throughput can be optimized for different networking environments and traffic types. The P1000 with PhobosLink software has a suggested retail price of $1,695 US.

Contact: Phobos Corporation, Commerce Park, 488 East Winchester Street, Suite 150, Salt Lake City, UT 84107, 801-474-9200, 801-474-9201 (fax), sales@phobos.com, http://www.phobos.com/.

RAIDION.fc Series Fibre Channel Disk Array

The Raidion Systems Division of Peripheral Technology Group, Inc. announced the RAIDION.fc series Fibre Channel disk array, a new addition to its family of fault-tolerant data solutions. Features include Linux support, an entry point of three-drive modules scaling to 90, fully redundant components, dual-active storage processors, dual-loop fibre-capable and mirrored write-cache. The rack-mounted Raidion FC disk array has a list price starting at $12,000 US.

Contact: Peripheral Technology Group, Inc., 7580 Quattro Drive, Chanhassen, MN 55317, 800-875-0068, sales@ptgs.com, http://www.raidionsystems.com/.

Security & Hacking 1.0 CD-ROM

SuperAnt announced a security CD-ROM containing sniffers, port scanners, PGPi, cryptography tools, hex editors, disassemblers, nuke protection Linux administration, programming guides and other tools useful for protecting systems. It costs $9.95 US and includes Linux and DOS-based solutions such as networking utilities, e-mail tools and FTP.

Contact: SuperAnt, 2531 Sawtelle, #102, Los Angeles, CA 90064, superant@superant.com, http://www.superant.com/.

maXimumcde

Xi Graphics, Inc. offers desktop and laptop users a powerful GUI with its new maXimumcde for Linux. The fully integrated package includes Motif and CDE

built on the Accelerated-X Display Server. Other features include fully integrated utilities; auto detection of graphics hardware at install; English, German, French and Italian language desktop support; graphical session and login manager; and overlay support on capable hardware. maXimumcde for Linux Executive Edition is available for $199.95 US; the Developer's Edition is $349.95 US. Laptop versions and international pricing are slightly higher.

Contact: Xi Graphics, Inc., 1801 Broadway, Suite 1710, Denver, CO 80202, 303-298-7478, 303-298-1406 (fax), sales@xig.com, http://www.xig.com/.

Quad Xeon System with Intel 550 Processors

Penguin Computing announced that it is now offering Quad Xeon systems utilizing Intel 550 MHz Processors. The computer was designed in response to a specific customer request, by a team of engineers at Penguin dedicated to developing the fastest and most reliable Linux systems available. Pricing of the Quad Xeon machines starts at less than $14,000 US.

Contact: Penguin Computing, 965 Mission Street, Suite 630, San Francisco, CA 94103, 888-736-4846, sales@penguincomputing.com, http://www.penguincomputing.com/.

TurboLinux Workstation 3.6

TurboLinux announced Workstation 3.6, shipping with the latest 2.2.9 kernel and the option of a GNOME or KDE desktop. It includes source code on two CD-ROMs, a companion CD-ROM with bonus Linux applications, developer tools and office productivity suites (including the download version of Corel's WordPerfect 8 for Linux), a 300-page manual and 60 days of free installation support. Price is $49.95 US.

Contact: TurboLinux, 2000 Sierra Point Parkway, Suite 702, Brisbane, CA 94005, 801-501-0866, 801-501-0889 (fax), orders@pht.com, http://www.turbolinux.com/.

Archive Index Issue Table of Contents

Advanced search

Java Servlets

Doug Welzel

Issue #66, October 1999

An introduction to writing and running Java servlets on Linux.

Java servlets are essentially Java programs which extend the functionality of a server. They are not confined to web servers, but are most often referred to in this context. Virtually all references to servlets cite them as a replacement for CGI scripts, so it is easiest to think of them as Java programs that perform CGI functions.

The intriguing thing about servlets is their claimed performance. Traditional CGI scripts written in Perl, C, etc. all have a disadvantage in that a new process must be created for each call of the script. The overhead of process creation and management can be very taxing on a loaded server. Servlets solve this problem by creating a thread for each request, rather than an entire process. A single process for each servlet is created, then a request to the servlet causes a thread to be created to handle it.

Sounds great, but how does one use servlets on Linux? Well, you need a web server that supports both servlets and a Java virtual machine. There are several choices in the web server department. Sun's Java Web Server will probably run on Linux (since it is written in Java), but it is commercial. We'll be using Apache in this article because it is free and is widely used. This means we need a servlet extension for Apache. Livesoftware makes a product called JRun which I have heard great things about, but again, we'll stick strictly to the Apache side of the world and go with their mod_jserv extension.

Next, you'll need to choose a JDK (Java Development Kit) for your system. Again, there are several choices. Two worth mentioning are the Blackdown JDK (http://java.blackdown.org/) and the new OpenGroup JDK (http://www.camb.opengroup.org/RI/java/linux/), which uses a native threads implementation. Since threads are important to servlet performance, your JDK choice might significantly impact performance. I used the Blackdown JDK,

because I was familiar with it and knew it was stable. However, the OpenGroup's work is worth looking into if you have the time.

To get servlets up and running on your system, follow the steps below. Note I assume the JDK you chose is installed and working.

The first step is to download the latest version of Apache (http://www.apache.org/) and the latest version of JServ (http://java.apache.org/). These were 1.3.1 and 0.9.11 respectively, at the time of writing.

The second step is to unpack each of the archives using the **tar** command:

```
tar -zxvf apache_1.3.1.tar.gz
tar -zxvf jserv0.9.11.tar.gz
```

The third step is to compile Apache with the JServ module. Specify any options you need in the Apache configuration, but make sure to include an **--add-module** switch to specify where the mod_jserv.c file is located. Here is a simple example of an Apache configuration command:

```
./configure --prefix=/usr/local/apache
              --add-module=/usr/local/src/
              jserv0.9.11/mod_jserv.c
```

This will automatically add the JServ module to your Apache configuration. Once configuration is finished, go ahead and make and install the package:

```
make
make install
```

The fourth step is to compile JServ. Before compiling JServ, you must pick a location for installation. If you are happy with the directory where you unpacked it in step two, then you are set. Otherwise, just move the JServ directory to wherever you want it to reside. To keep things clean, you might want to put it in your Apache installation tree.

Next, the **CLASSPATH** variable needs to be set. Even though the JServ documentation suggests this may not be necessary, I found the package didn't compile unless it was explicitly set. The **CLASSPATH** variable must specify the path to your JDK classes and the JSDK classes included in the JServ package. My JDK lives in /usr/local/jdk1.1.6 and the JDK classes archive is in the /lib directory of this tree. I have JServ in /usr/local/apache/jserv, so my **CLASSPATH** variable would be set as follows:

```
export CLASSPATH=/usr/local/jdk1.1.6/lib/\
classes.zip:/usr/local/apache/jserv/servclasses.zip
```

Once this is set, change to the JServ directory and compile it:

```
cd /usr/local/apache/jserv
make
```

Step five is to configure Apache. JServ requires that a number of configuration parameters be added to your Apache server configuration files. The files are typically located in the /etc directory of your Apache installation tree. Open up httpd.conf with your favorite editor and add the following configuration directives:

ServletBinary: the full pathname to your java binary. For example:

```
ServletBinary /usr/local/jdk1.1.6/bin/java
```

ServletClassPath: specifies the path to your various Java classes. JServ requires you to specify the path to the JDK, JSDK and JServ classes. For example:

```
ServletClassPath\
/usr/local/jdk1.1.6/lib/classes.zip
#path to the JDK classes
ServletClassPath\
/usr/local/apache/jserv/servclasses.zip
#path to the JSDK classes
ServletClassPath /usr/local/apache/jserv/classes
#path to the JServ classes
```

ServletAlias: this is one of the most important directives, since it configures the location of your servlets and how they are accessed. The syntax of the directive is:

```
ServletAlias uri directory_or_filename
```

where the ***uri*** argument specifies how your servlets will be accessed via URLs, and the second argument points to the actual location of the servlets. The second argument can either specify a directory containing the servlets or a ZIP/JAR file containing a collection of servlets.

For example, if your ServletAlias directive was

```
ServletAlias /servlets /usr/local/apache/servlets
```

then URLs addressing your servlets would look like http://*yourhostname*/servlets, and the actual servlets would reside in the /usr/local/apache/servlets directory.

ServletProperties: gives the location of a file containing properties for your servlets. The path can be absolute or relative to Apache's server root and, if not specified, defaults to

```
conf/servlets.properties
```

Many properties can be set within the properties file. Arguments can be passed to all servlets with the statement:

```
servlets.default.initArgs=arg1=val1,arg2=val2,...
```

Arguments can be passed to individual servlets as follows:

```
servlet.servletname.initArgs=arg1=val1,arg2=val2,...
```

The sixth and final step is to fire up Apache. Well, you should be ready to go at this point, so go into the /sbin directory of your Apache tree and start up the server:

```
cd /usr/local/apache/sbin
./apachectl start
```

Writing Servlets

The **Servlet** interface provides the foundation for all servlets. All servlets must either implement this interface or be extensions of a class which implements it. The Servlet package provides a class called **HttpServlet** which implements the Servlet interface, so as a servlet developer, much of the work is done for you. The Servlet interface allows the creation of generic servlets, but we will only look at how to create servlets that act as CGI scripts. For this, all you have to worry about is the **HttpServlet** class.

Listing 1.

Let's step through the code of the simple servlet shown in Listing 1 to see how it works. This servlet overrides the **doGet** method provided by the **HttpServlet** class. **doGet** is called when a client makes a GET request to the servlet. Here we have the servlet respond with a simple web page that gives the standard “Hello World” message. The doGet method gets two objects, **HttpServletRequest** and **HttpServletResponse**, which encapsulate information that allows the servlet to obtain information from and communicate with the client. For example, the **HttpServletResponse** object contains a **PrintWriter** object that can be used to send information back to the client. In this example, we use it to send our “Hello World” message back to the client. We also use the **setContentType** method of the **HttpServletResponse** object to inform the client that it will be receiving text/HTML data from the servlet.

Now that we've seen a simple example, let's step back a bit and look at how HTTP Servlets work. Servlets extending the **HttpServlet** class handle all of their client requests through its **service** method. The service method understands standard HTTP requests, and calls appropriate methods to handle each request. In the example above, the service would recognize the GET request and call the **doGet** method accordingly. Similarly, **doPost**, **doPut** and **doDelete** methods are provided to handle other types of HTTP requests.

The Life Cycle of a Servlet

A servlet's life begins when the servlet's **init** method is called. The web server calls the init method when it loads the servlet, but before any client requests are handled. The init method is called only once when the servlet is loaded. So, if you need to perform any initialization before your servlet starts handling requests, overload the **init** method as follows:

```
public void init(ServletConfig config) throws ServletException {
                super.init(config);

                ...
        }
```

The init method is passed an object which contains configuration information about the servlet. It is a good idea to store this object to make it available if the client needs it. The easiest way to do this is to call the init method of superclass and pass the **ServletConfig** object to it. One final tip regarding the initialization of servlets: if your initialization fails and the servlet can't handle client requests, throw an **UnavailableException**. After initialization takes place, the servlet is up and the service method handles client requests.

Finally, when the servlet is unloaded from the web server, its **destroy** method is called. The web server waits until all service methods are finished or a certain number of seconds have passed (whichever comes first) before calling the destroy method. In the case of long-running service methods, it is possible the destroy method will be called before all service calls have been completed.

This situation can be handled with a few additional methods and variables. First, create a variable that keeps track of how many service methods have been called and provide synchronized methods for increasing and decreasing the counter, as well as one to return the value of your service counter.

```
public MyServlet extends HttpServlet {
   private int numServices = 0;
   protected synchronized void enterService() {
   numServices++;
   }
   protected synchronized void exitService() {
   numServices-;
   }
   protected synchronized int serviceCount() {
   return numServices;
   }
}
```

Now, that we have these counters, we need to modify the service method to increment and decrement them accordingly. This is done by adding a call to the **enterService** method at the top of the service method, calling the service method of the superclass to handle the real work and then decrementing the counter by calling the **exitService** method.

```
protected void service(HttpServletRequest req, HttpServletResponse
resp)
   throws ServletException, IOException
   {
   enterService();
   try {
   super.service(req, resp);
   } finally {
   exitService();
   }
   }
```

Next, a flag is needed to determine if the servlet is in the process of shutting down. To accompany this flag, use accessor methods to set the flag and return its value.

```
MyServlet continued {
   private Boolean exiting;
   protected setExiting(Boolean flag) {
     exiting = flag;
   }
   protected Boolean isExiting() {
      return exiting;
   }
}
```

Now the destroy method should first check if any services haven't completed, then loop until all services are finished.

```
public void destroy() {
   if (serviceCount() > 0) {
      setShuttingDown(true);
   }
   while(serviceCount() > 0) {
      try {
      Thread.sleep(interval);
      } catch (InterruptedException e) {
      }
   }
}
```

Finally, modify any of your methods that may run for a long time to check if the servlet is shutting down, and act accordingly.

```
public void doPost(...) {
  /* You could do something like this or put
   * the check into a loop
   * that takes a long time */
   if(!isExiting) {
   ...
   }
}
```

So there you have it: a quick introduction to getting servlets up and running on your Linux box and writing some simple ones. If you want to learn more about writing servlets, books are available which cover them in depth. I would also recommend looking at the Java Tutorial, available on Sun's web site; it contains a nice introduction to servlets I used when I started learning about them.

Resources

Doug Welzel recently finished up his undergraduate work at Carnegie Mellon University and is continuing his graduate studies. He has been using Linux

throughout his career at CMU and welcomes your comments at welzel@andrew.cmu.edu.

Archive Index Issue Table of Contents

Advanced search

Bisel Bank

Pablo Trincavelli

Issue #66, October 1999

How a bank in Argentina is using Linux for testing database and web applications.

Bisel Bank was born around 1994 as a merging of many small banks. It is now one of the largest banks of Argentina with more than 160 branch offices (and plans to open many more) running in-house software, which will be replaced after the year 2000 with a commercial offering.

The central office runs on a Sun Enterprise 5500 server running Solaris 2.6 (with a similar one as a backup server), and all data is stored in an Informix Online 7 DSA database. The applications, written in JAM and ESQL/C, work quite well.

The branch offices use SCO UNIX and a mixture of Progress, ESQL/C and JAM/Informix applications. It all started when we had to consolidate about eight different systems from different banks. The JAM/Informix and Progress applications running on UNIX boxes won the battle against the other contenders, including some AS/400 hardware and software.

The software is developed and maintained by a group of programmers who continuously have to modify running programs or make new ones from scratch. We have to manage much “traffic” to and from the main server. To do this, we implemented a version control system for the programs using RCS (revision control system), and a system to send them to the main computer.

How This Works

After the requirement for a new program or change in an existing one has been met and the new or changed program is finished, the program passes through a set of testing and authorization stages before it moves into the production environment.

When all is okay, the programs are sent to the central office or to an automatic distribution system (as required), which sends the modifications to all the branch offices overnight. This system was implemented using **rsync** (on a Solaris server), so the amount of data transferred over the network is kept to a minimum.

Linux Entrance

Finally, Linux makes its triumphant entrance. It all began when I came to this bank as a formal employee in November 1997. Being a Linux user since kernel 0.99, I believed that Linux deserved its place in this bank scenario.

I decided to install a Linux server to use as a test box. First, I used this equipment to test several software packages, then when I was satisfied, I moved the software to the Solaris environment. SCO was out of the question for testing purposes, because it was the old SCO 3.2.4.2, which makes it difficult to port software for it.

I began using the Linux box to test products such as Samba, rdist, rsync, Apache Web Server, PHP/FI, PHP3, MSql, MySQL, Solid Server, Solid Web Engine, VNC, Squid and even Informix SE for Linux. Much of this software is being used now at the bank on either the Solaris, SCO or Linux platforms or a combination of them.

I implemented various projects, such as designing and implementing networks using Samba; RCS for source code; an Intranet for manuals, documentation and internal procedures; automatic distribution of applications using rdist, which was soon replaced with rsync to save transfer time; a couple of backup procedures over the network; and even some tests with Java and JDBC to access database servers.

Real Linux Work

One day, a new project came about: build an application to use the Intranet to send programs to the production environment. What I wanted was to have complete control over where a program is at every moment.

Figure 1. Program Management Entry Screen

First, the programmer writes a new program or a modification to an existing one using RCS (with a front end designed to ease the programmer's work) to keep control of the versions. Then he or she must tell someone the program is finished, so someone else can have a look at it and make the appropriate tests before passing it to the production environment.

This is done by logging in to the “Sistema de Pasaje de Programas” system and entering the name of the program. At this point, the program is ready to receive an authorization to be sent to the testing environment. Once the authorization is granted, it can effectively be sent to the testing environment.

After all the necessary tests are passed successfully, the program is ready to go to the production environment. This is done by a similar process. In the first stage, the program is left “ok” to be passed, so it requires another authorization, then the final pass to the production environment.

All these authorizations and passages are recorded in a database, so we can know exactly where one program is in every moment, i.e., when it has been authorized or passed to which environment.

All is done with a web-enabled application in which a record is inserted into the database, so the person in charge of the authorization finds it on a list on his screen when he checks whether there is something to be authorized. Then, this record is updated with the current user, date and time, so the person who makes the actual pass finds it on his list. It's easier to actually use than to explain in words. The application can also be sent to another testing environment with large quantities of data to make more extensive tests.

Figure 2. Login Screen

This system is hosted on a Slackware Linux server with 16MB of RAM, 1GB of disk space and a 133MHz Pentium processor. It has an Apache web server and a Solid database. It usually has an uptime of 60 or more days without any kind of problem. The HTML pages were made using a PHP3 build as an Apache module. This system was designed as a test for the Solid engine, which proves to be quite good—I recommend it. Because of the release of Informix SE for Linux and the use of Informix by our organization, I am reengineering the whole system with Informix SE or Informix OnLine, and it will be fully operational by the time you read this article.

Consideration

One interesting consideration in this system is the stage at which the program is actually passed to the test or production environment. This is done with CGI scripts which execute various commands directly related to our programming environment. The system can be adapted to a totally new programming environment by replacing only those CGI scripts—it is not JAM-dependent. It was created with this independence in mind, because the bank will change its system in the near future, so this independence guarantees we can continue to use it with minor adjustments on our new system.

Figure 3. Program History Screen

Under the Hood

The method I used to pass a program to a different machine, which runs Solaris using a web application, is simply installing a web server on the Solaris, then using a URL that references a CGI script on that remote machine. This CGI script is responsible for passing the program, issuing the necessary **rcp** command and any commands necessary to leave the program ready to be used.

As you can see, the actual work is done by CGI scripts and all the HTML pages are used to glue the scripts in a nice-looking, easy-to-use application which stores all the program "flow" between equipment and stages in a database. I easily added report pages to view the activities by day or to search by program name.

Current and New Projects

In addition to porting all this to Informix, we are currently developing an application for the Human Resources office to retrieve information on employees. This is being done in a similar way and will be hosted on this same Linux server.

Because of the robustness shown by this architecture, we'll be making more and more web applications in the future, and Linux will be there as our web server.

One Final Note

I am quite impressed by PHP3—this product is incredibly flexible and powerful and can handle complex applications without problems. Its database support is getting better, supporting not only the classic freeware and shareware databases such as mSQL, MySQL and Postgres, or commercial databases such as Solid, but also the big databases such as Informix, Oracle and Sybase.

Without any doubt, Linux has a wonderful business future and is my favourite OS for Intel machines, outperforming Windows NT and SCO UNIX. In my opinion, Linux and Solaris are the best operating systems on the market at this time.

One important aspect to consider is the type of technical support available for your OS and for any other product you regularly use. On one occasion, I was stuck with a problem (it was my fault) that forced the Linux server to go down. I received help in 20 minutes from three technicians. Where did I get this kind of excellent support? Of course, it was the Internet. I posted a message, and in 20

minutes my problem was solved. I have not seen this kind of fast response on any commercial product from any company.

Pablo Trincavelli works for Banco Bisel S.A. in Rosario, Argentina as a Technical Analyst. He has been working with Linux since the early 0.99 days. Other than Linux, he has also worked with Solaris, HP-UX, SCO UNIX, WinNT, AmigaOS and many others. He likes playing with his PalmPilot and finding easy ways to do difficult things. He likes everything with chips inside and can be reached at pablo@iname.com.

Archive Index Issue Table of Contents

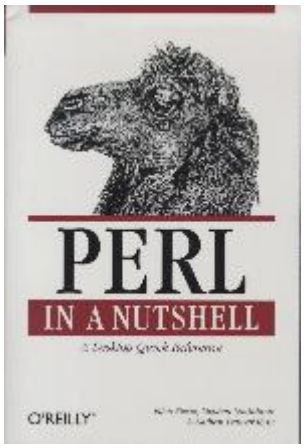
Advanced search

Perl in a Nutshell

Jan Rooijackers

Issue #66, October 1999

This book was hard to review, because it is a reference guide. One thing is clear: the authors have made it a good reference guide.



- Authors: Ellen Siever, Stephen Spainhour and Nathan Patwardhan
- Publisher: O'Reilly & Associates
- E-mail: info@ora.com
- URL: http://www.ora.com/
- Price: $24.95
- ISBN: 1565922867
- Reviewer: Jan Rooijackers

A quick stroll through the book gave me the impression that much time was spent putting it together. The book is divided into eight parts, each containing one or more chapters.

Chapter 1 is a small but very useful chapter. It explains how Perl can be used and which platforms support it. The most important paragraph in this chapter is “Perl Resources”. All well-known resources are mentioned here: newsgroups,

FAQs (frequently asked questions), web pages and other books from the publisher.

Now that you know how to get Perl, it is time you got it installed. Specific installation topics like installing modules and documentation are explained in Chapter 2.

In Chapter 3, the Perl interpreter and its options are explained from command-line to environment variables. Chapter 4 explains the program structure. This chapter and the following one are good references for anyone already familiar with Perl or another programming language. If you are totally new to programming and you want to learn Perl, the book *Learning Perl* (Randal L. Swartz and Tom Christianson, also from O'Reilly) might be a better place to start.

Chapter 5 gives brief descriptions of the built-in functions. Each description covers the syntax of the function, with the types and order of its arguments. Since no one writes perfect code all the time, the debugger is explained with its various options. Packages, modules and objects are discussed in chapters 7 and 8. A short explanation and an example give you a good idea of how each standard module, delivered with Perl, is used.

CGI is covered in chapters 9 through 11. I feel this section will be used the most, because CGI together with the Internet is such a hot item. Many web pages make use of CGI scripts.

Chapter 11 discusses the useful things which can be done by **mod_perl** as it merges Perl into the Apache web server. This module will reduce the forking of extra processes each time a script is executed. Installing this module and making use of its capabilities can speed up execution on the server.

Chapter 12 explains the connection between databases and Perl. Databases are also a hot item on the Internet these days. DBM (database management) and DBI (database interface) are covered in this chapter.

Network topics such as sockets, e-mail, news, FTP and LWP (library for web access in Perl) are discussed in part six. Each chapter starts with a short description of the service, followed by the key words in alphabetical order along with their uses.

Perl/Tk is discussed in Chapter 18. I did some design work with Tcl/Tk in the past, and was amazed at how easy it was to get something done. With Perl and Tk, it is even simpler. Using the examples, you can quickly design some nice things.

The last chapter covers Win32 modules and extensions. With the information provided in this chapter, you could create some nice things in the Microsoft world.

The index may be the most important part of a book, in my opinion. A bad index will cost you time when searching for topics in a book. This index is so complete, you will easily find any desired information.

This book was hard to review, because it is a reference guide. One thing is clear: the authors have made it a good reference guide. This book is a must for people who already know Perl and want to learn more. The information you need is easy to find and understand due to clearly written text and good examples.

Jan Rooijackers works for Ericsson as an Internet Consultant. He has been in contact with Linux since 1994 and thinks it is nice to see how popular it has become—even Windows is afraid of it. He spends his time with his wife, two sons (Mike and Brian) and his computer. He can be reached at Jan.Rooijackers@dsn.ericsson.se.

Archive Index Issue Table of Contents

Advanced search

Java 2 Software Development Kit

Harry J. Foxwell

Issue #66, October 1999

The Java 2 SDK is mow available for Linux. Mr. Foxwell tells us all about it.

Since its introduction four years ago, Sun's Java technology has appeared on nearly every size and type of computer system, from IBM's MVS-powered mainframes to 3Com's Palm organizers.

New versions of the Java Development Kit and Java Virtual Machine are available first on Sun's systems and on Microsoft Windows, but many months may pass before they appear on other platforms. Sun's Java Software division produces reference implementations *only* for Solaris and for Windows, and licenses source code to other system vendors who want to port the JDK or JVM to their own hardware and operating systems.

Some vendors, like IBM, have their ports of new Java versions ready almost immediately. Unfortunately, users of platforms such as Apple's Macintosh have had long waits for the latest JDK. Linux users didn't see JDK 1.1 until mid-1997, although Sun introduced it in December 1996, and they have been eagerly awaiting Java 2 since November of last year.

Sun had no problem figuring out with whom to arrange Java licensing agreements at the major system vendors, but it did take them a while to identify a licensee for the widely dispersed Linux community. They eventually licensed the JDK source code to Steve Byrne and the Blackdown Java-Linux Porting Team, a group of Linux developers.

The Blackdown team recently released a preliminary version of the Java 2 SDK aft er testing it extensively with the Java Compatibility Kit, a suite of more than 16,000 tests which verify a port's conformance to the published Java language and runtime specifications.

You can download the Java 2 SDK for Linux from the Blackdown mirror web sites, found at http://www.blackdown.org/. Note that Sun recently changed the name of the software from JDK 1.2 to Java 2 SDK, although much of the on-line material still refers to the earlier name. The compressed SDK is nearly 24MB, so use a fast network connection if possible. Full documentation for Java 2 is packaged separately from the SDK. Download it directly from Sun's web site at java.sun.com/products/jdk/1.2/docs/index.html.

Installation and Testing

Installation consists of uncompressing and extracting the jdk1.2pre-v2.tar.bz2 file archive into your home directory or other location. Use the command:

```
bzip2 -d jdk1.2pre-v2.tar.bz2
tar xvf jdk1.2pre-v2.tar
```

In order to use the JDK, you must have a Linux distribution that includes the glibc2 version of the GNU C library. Most distributions, such as Red Hat 5.2, include this library, which is also called libc6.

After installation, simply set your **PATH** environment variable to include the JDK's bin directory, and you're ready to go:

```
export PATH={JDK-install-directory}/jdk1.2/bin:$PATH
```

To verify your installation, type:

```
java -version
```

The JVM should run and report its version as “java version 1.2”, along with additional information about the implementation.

The Java 2 SDK includes several tools for development and testing, including the **javac** compiler, the **jar** Java archive manager that works similarly to the **tar** command, and the **appletviewer** program for testing applets. “Applets” are Java programs intended to run within a browser; Java “applications” are like any other program and don't need a browser to run. Current versions of Netscape and Internet Explorer do not directly support Java 2, so you will need to use appletviewer to test your applets.

The Java 2 SDK comes with a compiler, but does *not* include an interactive development environment. You can, of course, use **vi** or **emacs** to create your source files, then compile and run them from your command line. For example, create the file HelloLinux.java containing the lines:

```
public class HelloLinux
   { public static void main(String[] args)
      { System.out.println("Hello, Linux!"); } }
```

Compile your program with the command **javac HelloLinux.java**. The compiler will produce the file HelloLinux.class, which you run using the command **java HelloLinux**. If you encounter problems trying to compile and run Java programs, check your **PATH** and **CLASSPATH** environment variables. **CLASSPATH** lists the directories where the JVM looks for class files. If your own files or the class files required for a separately installed Java library are not referenced in **CLASSPATH**, the JVM cannot find and run them.

Major New Features of the Java 2 SDK

The Java 2 SDK includes the Java Foundation Classes, also known as the “Swing” class library. This library extends Java's original Abstract Window Toolkit, used for building graphical user interfaces. The Swing library provides a rich set of components for GUI development. The jdk1.2/demo/jfc directory contains more than 100 example source files that demonstrate the library's capabilities, including the **SwingSetApplet** demo, which shows *all* the components at work in one big applet. The SwingSetApplet includes full source code.

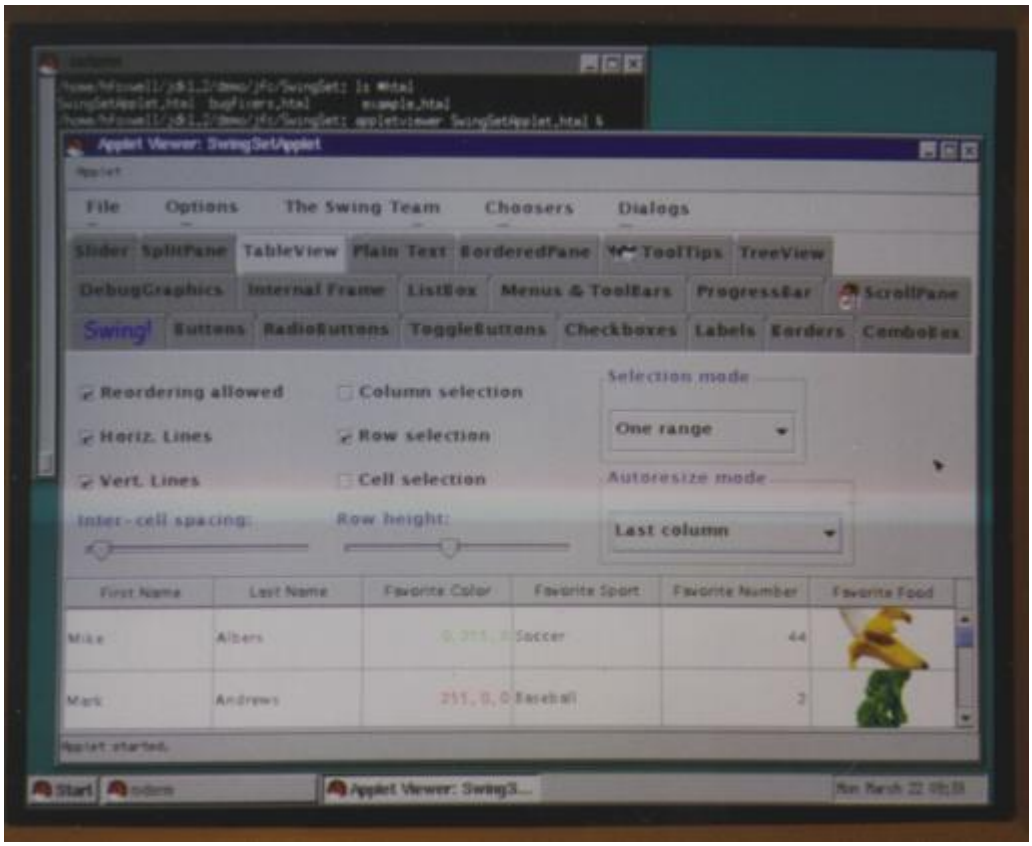


Figure 1. SwingSetApplet

Look in the jdk1.2/demo/jfc/Java2D directory, and you'll find the remarkable **Java2DemoApplet**, which shows off Java's 2-D imaging APIs, used to control image rendering, fonts, animation and printing.

One of the most significant changes to the Java platform concerns security. Earlier versions treated Java applets and applications differently, allowing applications unrestricted access to any system resource, while confining remotely loaded applets to an all-or-nothing “Sandbox” model. An applet in the old security model could not access client system resources and could not connect to any system other than its originating host. This provided a strong security model for remotely loaded executable code, but limited an applet's utility.

Java 2's new security model is based on a configurable policy file that can define security *domains* for each user, group and program component. This file, called the codeBase, tells the JVM where programs must be loaded in order to execute and determines who can run the program using digital signatures to authenticate the program's origin.

You can find a sample global policy file for all users in the jdk1.2/jre/lib/security directory. This file may be customized, and the JVM can be directed to use the policy file for all code execution permissions. An especially important point to note is that programs need not be modified in order to control their execution; the JVM and policy files determine what can run and who can run it. Take a look at http://java.sun.com/security/ for details and examples of Java 2 security.

Java Programming on Linux

If you are looking for professional Java 2 program development tools for Linux, you are again at the mercy of the tool developers, who generally release their products for Windows first and for other platforms later. However, if the development tool is written in 100% Pure Java, you are in luck. Blackdown's *Java Tools for Linux* link lists more than a dozen Java IDEs, including several that have been updated to work with Java 2. Check out NetBeans at http://www.netbeans.com/ and Together/J at http://www.togetherj.com/ for two excellent programming tools.

Other Java APIs

Sun has licensed source code for additional APIs to the Blackdown team, including code for Java 3-D, Java Media Framework, Java Advanced Imaging and Java Sound. Furthermore, Sun is now making sources available for much of the Java core technologies, including the new Jini software, under its Community Source Licensing Model. While not quite the same as the GNU General Public License, this model should make it easier for future ports of Java APIs to Linux and other platforms.

Linux developers are already at work adapting their solutions to Java 2. For example, Douglas Lau and Trent Jarvi have implemented the Java

Communications API, used for access to serial and parallel ports. You can download the necessary libraries from http://jarvi.ezlink.com/rxtx/.

Learning the Java Programming Language

If you are new to Java programming, your background will determine how quickly you can learn it. Programmers already experienced with object concepts through C++ or Smalltalk can usually become proficient in Java after a few weeks of study and practice. If you are an “object challenged” C or FORTRAN programmer, you will need help learning to think and design using objects; taking a course in object-oriented programming will get you started. Beginners should study Sun's *Java Tutorial* or Bruce Eckel's *Thinking in Java*, while experienced object programmers can jump right in with *Core Java* by Horstmann and Cornell.

Conclusion

I can't think of a better platform and user community for Java than Linux. Now that Java 2 is available, with its advanced GUI features and flexible security model, I anticipate a major wave of new and interesting Java programs. The Blackdown team deserves applause for its efforts.

Resources

Harry Foxwell is a System Engineer for Sun Microsystems. He consults with Sun customers on Java technologies and solutions; he also maintains Sun's internal web resource of Linux information. In his “spare” time, he is pursuing a doctorate in Information Technology at George Mason University. You can reach him at harry.foxwell@sun.com.

Archive Index Issue Table of Contents

Advanced search

LIMP: Large Image Manipulation Project

Valient Gough

Issue #66, October 1999

Designing a new library for processing large images using a minimal amount of memory.

Library design is an imprecise art. It would be impractical to include in a design all possible uses for a library of any sufficient size or complexity. Thus, it is inevitable that many libraries (and programs) reach evolutionary dead ends, where newly anticipated uses or algorithms can no longer fit nicely into the existing architecture.

The quickest short term solution is usually to hack in new interfaces, but over time the accumulation of hacks tends to reduce the stability, understandability and maintainability of code. The software industry comes up against this problem often, but fortunately has found a simple yet elegant solution—start over. Usually, rewriting lower- to middle-level interfaces can remove the accumulation of hacks. However, if the design criteria in question are incorporated throughout the code, sometimes only a total rewrite can truly help.

This is not solely a trait of software design. Software engineering is an expression of mathematics in a confined space (your computer). Through this heritage, it shares traits with other inexact sciences such as physics, where rewriting or reworking theories is not uncommon.

Starting Over

During the last five years, I've written many image-processing algorithms, from specialized routines for machine vision to complete libraries for commercial video and aerial image-processing software. The last commercial library has been in use for three years and has weathered many interface changes and enhancements. But just as each library was in some ways an improvement over previous attempts, I saw ways to improve performance and capability.

Following in the footsteps of the makers of the six million dollar man—I wanted to make it faster, smarter and better than before and at significantly reduced cost.

My commercial library had a large amount of code tied to it, so simply modifying the existing code was not an option. It seemed as if I would never be able to incorporate a new library into my commercial work because of enormous design incompatibilities. Rather than have this new library be destined to collect electronic dust on my hard drive, I decided to start completely from scratch as open source. In late November 1998, the Large Image Manipulation Program (LIMP) was born.

It's likely that even as open source, this library would have been inconspicuous enough to draw little, if any, attention. However, after a few months spent developing LIMP in my spare time, Open Source Remote Sensing (OSRS, http://remotesensing.org/) was born. I was thrilled at the thought of having an open source library that was actually useful to someone, so LIMP was moved to OSRS for public development.

Speed, Ease of Use and Memory

The purpose of LIMP is to allow the processing of large images using a minimal amount of memory. A number of available libraries can be used for image processing, any of which could be used to give identical results. The differences between these libraries can often be summed up by answering a few questions:

- Can the data be processed on demand, or must all the processed data be in memory or on disk?
- How easy is it to write new algorithms for the library?
- How efficient is it?

The simplest image processing library would first allow loading an image into memory, then provide pixel-level access to the data (read and write), and finally allow storing the data back to disk. Advantages of such a scheme include a simple set of interfaces and (given enough memory or small enough images) nearly optimal computational efficiency. Disadvantages include high memory usage and therefore poor scaling with image size or number of images.

If memory usage is not a problem, this would be the optimal way of dealing with images. Unfortunately, memory cannot yet be considered infinite for many image-processing demands. As an example, the very first test of my last commercial library was to load 1200 images consisting of 300 gigabytes of data and display them at once. Actual processing was done in blocks of images to avoid having a failure wipe out weeks of processing time. In an attempt to avoid

repeating historical blunders, I would never say that no one will ever have several hundred gigabytes of memory. I imagine when that time comes, people will work with even larger data sets.

In handling large images, many proprietary libraries reduce memory usage by sacrificing ease of use and efficiency. Great lengths can be taken to reduce the loss, but all such libraries are at a slight disadvantage in these areas and LIMP is no exception.

By learning from past experiences, many interfaces in LIMP have been designed to promote speed-enhancing optimizations as well as to group complex code into a few internal locations, where they can be more easily maintained. Because complex code is grouped into reusable templates, many types of functions and conversions can be written without having to deal with any complexity that would normally be encountered in large image processing.

Design Decisions

LIMP is almost pure C++, as is most of the code I've written in the last several years. My feeling is the compiler should do as much of the work as possible, allowing the programmer to focus on higher-level concepts. C++ is certainly not the holy grail of programming, but I feel that using an object-oriented language is much simpler then manually approximating an object-oriented interface in a non-object-oriented language. After all, everyone knows all computer languages can be reduced to assembly language; it is just a matter of how much work you have to do vs. how much the compiler does for you.

The Qt library (www.troll.no) is used as both a widget set and a template library. Since all of the core library is independent of the display subsystem, a library such as STL could have been used, but Qt is better documented with no inconsistencies between platforms. I also intended to write graphical interfaces using the core libraries, so Qt was a natural choice.

Plug-in types are used for a number of interfaces. This makes it easy to add new implementations without changing existing code. Image loading, saving and serialization are accomplished using plug-in interfaces. Simple interpolation filters are also implemented this way. The plug-in manager is very generic and can handle plug-ins of any type. This also makes it possible to add run-time loading of external plug-ins, although this feature is not yet implemented.

An image in LIMP is nothing more than a class for caching and moving data. By itself, an image does not produce, or in any way modify, the data. All production and processing steps are performed in “layers”. An image can contain any number of layers, but the minimum for useful work is one—the

source layer. This layer normally corresponds to some type of file loader (e.g., tiff), but can also be a simpler type such as an in-memory buffer, a constant-value image, or anything that can produce an image from scratch. In order to deal with large images efficiently, all data should be produced or loaded on-demand.

Other layers could perform such functions as data format conversion, radiometric or geometric transforms, and mosaics by combining multiple images. A number of data-type conversions are predefined (e.g., from RGB to YCbCr or RGB to gray-scale), and many other conversions can be easily defined. When processing layers are added to an image, everything about the image can be affected. The 2-D properties (width/height) can change, or the depth (samples per pixel, pixel data type, etc.) could be modified. For example, one class that modifies 2-D size is the zoom layer, which produces a new virtual image that is a magnification of an existing one. This can be used not only for zooming in on a visual image, but for up-sampling nearly any supported type.

By design, as much processing as possible is moved into an assembly-line approach. The basic unit for loading and processing data is a tile. All requests for data are made to an **Image** class, where it is broken up into tiles for processing. By determining which tiles are needed ahead of time, additional optimizations can be performed (for example, reordering the tile requests to optimize data cache hits). Each tile is processed and the ones not in the cache are created. All the complexity of chaining together layers of various types is dealt with by the **Image** class, which simplifies layer construction. When the time comes for a layer to process a given tile, it is presented with the input data space already filled and the output data space already allocated; therefore, it only has to process the pixels.

Current Status

Linux is the primary development platform; however, efforts are being made to keep the library portable to other platforms. The GNU auto-configure tools are used to test for required system characteristics. LIMP is known to build on Red Hat/ix86 5.2, Irix/Mips 6.3, and Red Hat/Alpha 5.2—each using **egcs** 1.1.2.

Support exists for tiff images in a variety of formats; scan-line tiff, as well as tiled tiff, are supported. Color and gray-scale capability is known to work, but the tiff layer also supports a number of other formats including shorts, floats and doubles, as well as multi-channel types. Recent work by others, notably Frank Warmerdam, has resulted in support for other image formats by bridging to his open source GDAL library.

Many optimizations that have been designed for LIMP are not yet fully realized. This is not to say that LIMP is slow in its current state, but room for

improvement exists within design specs. Already, some optimizations have been added which could be difficult to add in other architectures. For example, LIMP implements data-request optimizations that reorder requests to make optimal use of the image cache. Most of the optimizations are done internally, so the calling objects benefit from the optimizations without adding any extra complexity.

Today, LIMP is meant primarily for developers. The only thing usable for non-programmers is its image viewing program (**imgview**). **imgview** is a simple display tool, capable of viewing very large images quickly and without needing much memory. It takes user interface ideas from other image-processing tools to allow options such as dragging the canvas while updating the display in the background. It also supports a number of zoom filters for smoothing upsampled (i.e., enlarged) image data.

Future Work

What LIMP does now is only the tip of the iceberg. The basic foundation has been laid, but many algorithms remain to be written. Every area of LIMP will most likely receive extensions as work progresses. Of course, the people who contribute to LIMP and OSRS are the people who will drive the development. I will outline some of the planned work to give ideas of the kinds of things I believe would be useful. This is not an all-inclusive list, as there are likely many interesting features which have yet to occur to me.

Support for geographic information needs to be added at some level to allow applications to learn how images are related to the real world and each other. This is a basic requirement for high-level GIS programs. It may not be necessary to put this information directly into LIMP, but the metadata information in an image provides a potentially convenient way of storing and retrieving such information.

The classes from LIMP's image viewer will be extended to handle multiple overlapping images as well as vector data. This is also somewhat detached from the core of LIMP, because it would create a new display pipeline. The image display class already consists of a sophisticated drawing class, which is capable of ordering and computing tiles for the display with minimal impact on the GUI. This impact can be reduced to almost nothing once Qt supports threaded event handling.

A wide variety of radiometric image adjustments will be useful. These will start as simple histogram stretches for viewing, and progress to more complex color and intensity modifications. This type of modification should add very little extra overhead to LIMP, as it was designed specifically to minimize the procedural and computational overhead of such objects.

For a more complete list of expected modifications to LIMP, see the TODO file in the LIMP distribution. Similarly, if you are interested in the progress, see the NEWS and ChangeLog files.

Target Audience

As with most libraries that cater to processing extremes, LIMP is not destined for a mass-market audience. Good solutions for dealing with similar image-processing demands, such as editing, already exist. The problems facing a designer of a general editing program, where every pixel may be changed interactively, are not driving our choices. Instead, LIMP is designed to deal well with scientific image-processing needs. In this category, I am most familiar with aerial and satellite image processing, but I imagine other fields have similar needs.

Images for the GIS market typically cover a large area with a relatively low image scale. One obvious potential parallel in another field would be small area images with higher image scales—as might be found in microscopy work. As everyone who has played with a fractal generator knows, if you zoom into an object far enough, the original object seen at that scale covers an incredibly immense area.

Aerial and satellite images have come to be processed and stored on computers only in recent history. As computers become more powerful and capable of accessing larger amounts of data, new attempts will undoubtedly be made to process and understand exponentially larger sets of data at even finer resolutions. LIMP is not expected to be a final answer to these problems, but is just an experiment in dealing with them while maintaining performance, ease of use and the sanity of its programmers.

Valient Gough is the Director of Product Development at Stellacore Corporation in Denver, and is a long-time Linux user. When not programming in a sunlight-deprived area, he can be found skydiving at Colorado drop zones. He can be reached at vgough@remotesensing.org.

Archive Index Issue Table of Contents

Advanced search

Web Client Programming Using Perl

Robb Hill

Issue #66, October 1999

Web site monitoring of your system can be easy using Linux and Perl.

Many users of Linux are initially attracted to the platform for its powerful web server capabilities. Developing a web site on the Linux platform is a very satisfying experience. After the fun of designing and developing the site is over, it moves to production. Once in production, a web site typically has uptime requirements; to ensure these are met, the site must be monitored. I use a monitoring system that requests pages from a number of web sites, several times a day. If any server does not serve a web page, it is retried. If again there is no response, I am alerted via pager. Using this system, I can respond to an outage even before a user can report a problem. It takes only a little Perl and Linux to make this happen.

Linux and Perl were the obvious choices for a web server monitoring and alerting facility. Linux provides the stability and flexibility in the platform, and Perl, using the LWP bundle of modules (also known as libwww), provides an excellent means to work with HTTP requests.

I will explain how to compile/install the components needed, how to create a simple Perl script that will “HTTP ping” a server (HTTPping.pl), how to send a page to a pager using Perl, and how to glue it all together into an industrial-strength web site monitoring solution (Monitor.pl).

LWP

The wide availability of modules that extend Perl's functionality is one of the language's strong suits. In the case of fetching web pages, there are several modules that could be used. LWP is the clear choice, because it is a fantastic set of modules giving Perl powerful control over HTTP and HTML. Before you can begin exploring LWP and how you can use it to monitor a web site, you must install it.

Table 1.

The LWP modules depend on several other modules. These may not be installed on your system. If not, they must be downloaded and installed. Table 1 contains the exact version I installed, in the order I installed them. Except for SSLeay (more on that later), you can download all these packages from CPAN (Comprehensive Perl Archive Network). If you are new to Perl, it would be a good idea to browse around CPAN for a bit. It contains an enormous number of support modules for Perl that can save a lot of programming time and costs.

Once you have downloaded the tar.gz files, they must be expanded using the **tar** command:

```
tar -zxvpf MIME-Base64-2.11.tar.gz
```

This will uncompress and unpack the package. Do this for all archives. When finished, you will have a directory full of the original archives and a new folder for each archive. Each folder contains the module's installer; run the installer for each package. Again, using the MIME::Base64 module as the example, change (using **cd**) to the HTML-Parser-2.22 directory and type the following commands:

```
perl Makefile.PL
make
make test
make install
```

This is the typical way of installing Perl modules. However, packages vary, and you should always read the README or INSTALL file and other documentation that comes with any module.

SSLeay

If you plan to HTTP ping servers that use SSL (secure sockets layer), you must compile and install SSLeay and Crypt-SSLeay. SSLeay is a set of programs that provides the cryptographic routines needed for SSL. Crypt-SSLeay is the Perl module that serves as a wrapper for SSLeay. I suggest you read the documentation that comes with SSLeay. It will help especially if you run into any problems when compiling.

Execute the following commands in the directory to which you uncompressed the SSLeay-0.6.6b.tar.gz archive to install SSLeay:

```
./Configure linux-elf
make depend
make
make rehash
make test
make install
```

Three important things about SSLeay are:

1. Don't try to use the newer versions of SSLeay. They will not work with Crypt-SSleay and the LWP bundle. The last stable version is SSLeay-0.6.6b.tar.gz.
2. Second, SSLeay leaks memory. If you choose to use SSLeay and LWP to repeatedly HTTP ping a site that uses SSL, you will probably experience some problems with memory leaks. By calling HTTPping.pl from Monitor.pl, the script executes, pinging the site once, then terminates; this reduces the potential for a memory leak to accumulate, thereby causing your application to crash.
3. Since SSLeay uses RSA, you should read the SSLeay FAQ (see references) to determine if you can legally use SSLeay and if you need a license from RSA.

Creating an HTTP Ping Utility

HTTPPing.pl sends a HEAD request to the web server to be monitored. The server responds by sending only the HTTP headers. If you are testing only if the server is up, this is all you need to know. It is a good idea to understand the basic HTTP methods, GET, POST and HEAD, when beginning web-client programming. To see how the HEAD command works, we can emulate a browser using **telnet**. I have a web server running on the machine that I use for development. If you do not have a server running on your local machine, replace **localhost** with the name of a production web server.

```
$ telnet www.cpan.org
80Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
```

Type the following, then press “enter” twice:

```
HEAD / HTTP/1.0
```

This is the output returned by my server.

```
HTTP/1.1 200 OK
Date: Fri, 21 May 1999 03:18:26 GMT
Server: Apache/1.3.6 (Unix) (Red Hat/Linux)
Last Modified: Wed, 0 1999 21:17:54 GMT
ETag: "41803-799-370bcb82"
Accept-Ranges: bytes
Content-Length: 1945
Connection: close
Content-Type: text/html
```

This block of text contains the HTTP headers for the requested file. It can be automated using Perl and LWP as shown in Listing 1.

Listing 1. HTTPPing.pl

The first line of the script determines which interpreter to use to run the script. If Perl is in a different location on your system, you will need to change your script accordingly. HTTPPing.pl accepts two arguments: the URL to HTTP ping and a debug flag. These arguments are placed into variables. **LWP::UserAgent** and an **HTTP::Request** objects are created. The Request is passed to the UserAgent, and a response is returned. The response is examined to see if the server is up. If the site is up, the script exits with a status of 1; if not, it exits with a status of 0. Optionally, determined by the debug parameter, the status of the site is verbosely stated as “up” or “down”.

To ping a site, enter the command:

```
$ perl HTTPPing.pl http://localhost/ 1
http://localhost is up.
```

If you have a site that uses SSL and a user name and password, you could use:

```
$ perl
HTTPPing.pl https://username:password@localhost/ 1
https://username:password@localhost/ is up.
```

Creating the Paging Subroutine

Now that you can HTTP ping your site, let's see how to send the alert page. Once that is accomplished, you can tie HTTPPing.pl and the SendPage subroutine together by calling them both from a Perl script running as a service.

Several options are available for sending a page from a Perl script: the Simple Network Paging Protocol (SNPP), TAP (telocator alphanumeric protocol) and proprietary paging interfaces. Many pager services allow SNPP access. Also, a Perl module, Net::SNPP, can make sending a page very simple.

My paging service provides a web-based paging interface. It is an HTML form which collects the recipient's pager PIN and the message, and puts (POST) that information in a CGI script at my paging service provider. Since this is a web-based interface, LWP is a natural for the job. The form requires three values to be posted: the command type, the PIN and the message. The command type is constant for sending pages to my type of pager; the other two values are passed in as arguments.

Listing 2. SendPage Subroutine using Skytel Web Site

Note that the request object allows you to create a POST request as well as the already familiar HEAD request. This subroutine is the one I use in my Monitor.pl script. Most paging service providers have web-based paging services (see Resources).

Alternately, if your paging service provider gives you access to an SNPP server, consider using that protocol. Part of the libnet bundle is the Net::SNPP module. It is a great tool for sending pages using the SNPP (RFC 1861). It also has the appeal of being open-standards-based. However, some SNPP servers are so slow that many times, the above method will be superior. Using the Net::SNPP module to send a page is shown in Listing 3.

Listing 3. SendPage Subroutine for SNPP Servers

Tying It Together

Integrate HTTPPing.pl and whichever SendPage routine you chose. Monitor.pl repeatedly calls the HTTPPing.pl script at regular intervals. If HTTPPing.pl detects that the site is down, it retries the site to verify. If the site is verified as down, a page is sent. After sending the page, Monitor.pl continues to send HTTP pings to the site. When the site comes back up, another page is sent.

Listing 4. Monitor.pl

Below is an example of starting Monitor.pl, giving the URL to ping, the pager pin and the delays as arguments. The last argument is an optional debug flag that will show you what is going on inside the script.

```
perl Monitor.pl http://localhost 1234567 120 120 1
```

Once Monitor.pl is started, it will patiently test your site at an interval you choose for as long as you like. It is a good idea to start such a script during system boot.

Starting Monitor.pl at System Boot

Copy HTTPPing.pl and Monitor.pl to the /bin directory, and change their permissions to make them executable. This permits them to be run by the system at system boot.

```
chmod +x Monitor.pl
chmod +x HTTPping.pl
```

You can start the Monitor.pl script when your system boots by taking advantage of a special script that is run every time your system boots. The /etc/rc.d/rc.local script is run by the system after all services have been started. By adding the following line to the end of your rc.local file, the Monitor.pl will be started at boot time.

```
Monitor.pl http://localhost 1234567 120 120 0 &
```

Then use a tool such as **tksysv** to add it to the run levels you want it to start in. On my system, I boot to run level 3, so I added it to the list of the programs to start in run level 3. This makes the monitoring station even more robust by enabling the monitoring scripts to start up at boot time. See *Linux Journal* issue 55 for a good introduction to init and services.

Conclusion

The solid server performance of Linux and the rich and easy-to-use features of the LWP bundle and Perl would make it very easy to extend HTTPPing.pl and Monitor.pl to perform more complex web server checks. You could handle cookies and perform simulated user input into forms periodically to test applications. The possibilities of HTTP client programming are limitless with the LWP modules.

Resources

Robb Hill is the Senior Webmaster at the American Red Cross National Headquarters, where he is responsible for the technical aspects of the public and Intranet web servers. If you have questions, he can be reached at vortextube@earthlink.net.

Archive Index Issue Table of Contents

Advanced search

Open Source Software for Real-Time Solutions

Charles Curley

Issue #66, October 1999

Mr. Curley takes a look at two open-source solutions for embedded systems: RTLinux and eCOS from Cygnus.

Much hype is circulating in the industry today about open-source software and Linux. These discussions have focused on how Linux represents the first major threat to Microsoft's domination of the desktop. However, open-source software is also moving into the embedded real-time marketplace. Last year, Cygnus Solutions unveiled a new initiative called eCos (embedded Cygnus operating system). eCos is available as both open source and royalty free. Today, eCos has been downloaded by well over 10,000 developers. Why didn't Cygnus choose Linux as its open-source real-time alternative? This article will compare and contrast these two alternatives and explain why each can have a place in your real-time solution.

Cygnus and the GNU C Compiler

To see why Cygnus developed its own real-time operating system (RTOS), you must look at what is on the market and the type of job Cygnus wanted to take on with its RTOS.

Cygnus was founded in 1989, effectively pioneering the open source business model. While Linux is the poster child for open source today, Linus Torvalds was in fact inspired by earlier open-source projects, especially the GNU C compiler (**gcc**). The Free Software Foundation's (FSF) Richard Stallman first released the GNU C compiler in mid-1987, and Michael Tiemann, then a research scientist, quickly adopted it. Tiemann's contributions to the GNU C compiler, which included ports to several RISC and CISC microprocessors, a complete native-code C++ front-end, and optimization passes for instruction scheduling and branch scheduling, created more demand for code development and support than he (or any of the other gcc hackers) could provide in their spare time. Tiemann decided that this demand was not a fluke,

but a result of deeply seated economic principles, and founded Cygnus to prove it. (See Michael Tiemann, “Future of Cygnus Solutions” in *Open Sources: Voices from the Open Source Revolution*, O'Reilly, 1999.)

Cygnus was in the real-time business almost from the beginning. When the GNU C compiler acquired a reputation for runtime speed and compact code, it came to the attention of real-time developers. Exercising the freedoms uniquely available from the open-source bazaar, these savvy developers asked for, and contributed back, further enhancements to the compiler, making it an even better choice. Cygnus, their customers and the Net community were seeing the power of the bazaar in action.

Evolution and Configuration

Early on, Cygnus identified configuration tools as one of the key technologies for making it possible to support many microprocessors from a single source base. They created the “configure” script (one half of the famous “**configure; make**” process for building FSF and Linux applications and utilities), opening the doors to a true multiplatform solution. Many companies that depend on embedded processor projects move rapidly from processor to processor to gain new capabilities or to get the same capabilities in a smaller package. As a result, the GNU C compiler supports some 50 processors today, and 125 host-target combinations are available for cross compilation. In two years, Cygnus may see a 90% rollover in the processors it will support with gcc. They may write 24 new processor ports in a year, a tremendous output for a company that doesn't even own the product it is supporting.

Meanwhile, Cygnus assiduously sought out its customers' wants and wish lists. They had to understand why they were getting—or losing—sales. By 1995, they knew their customers needed standardized runtime support that integrated completely with the GNU gcc compiler, gdb debugger and tools on the host. The runtime support needed to include a C library, hardware abstraction layer (HAL), debugger support and a wide range of RTOS functionality. As with gcc, portability was a key ingredient, as it had to be available for a wide range of architectures and evaluation boards.

RTOS Considerations

In moving into the RTOS business, Cygnus had to consider its own strengths and weaknesses. To judge by the plethora of proprietary closed-source RTOSs (over 120 in 1995), anyone can make a buck with a proprietary RTOS. At least, that's what many people would like to think. Actually, an RTOS, like a restaurant, is a truly good way to go broke fast, unless you know exactly what you are doing.

What Cygnus needed was an open-source RTOS. Many products actually come quite close. Typically, the user has access to the source but the vendor owns it, and you don't see the source until after you buy the product. Part of the purchase is a non-disclosure agreement, which prevents you from trading improvements with other users of the same product. These constraints are anathema to the open-source model.

A number of Linux variants offer real-time capabilities, all of which are open source. Probably the best-known is Victor Yodaiken's RTLinux. More are listed at the Linux Embedded web site. Any RTOS must be highly configurable, because embedded projects run on very different hardware, almost always custom hardware. If you think PCs vary greatly, you haven't looked at embedded projects.

To give an example, Linux might support an infinite number of some resource —say, mutexes or timers—or it might supply a fixed but very large number of them. It must provide that flexibility, because the designers have no idea which applications the user will run. That flexibility has a cost: the resources, such as memory, to support it, or the code to spawn a new one. An embedded project designer knows exactly how many of something the project will need, so he can select that number before compiling. The embedded project trades runtime flexibility for compile-time flexibility and gains simplicity and reliability. Thus, in addition to open source, Cygnus needed an easily configured RTOS.

Offering real-time capabilities isn't the same as offering a full-blown RTOS. To see why, let's look at real-time programming and projects.

Real Time vs. Real Time

Dean Koester, Cygnus' Director of Product Marketing, sees a whole spectrum of real-time applications. They vary throughout that spectrum according to their requirements. We will look at Koester's spectrum by examining the two ends, and see why RTLinux and eCos fall where they do on the spectrum. While we are at it, we will look at the economic implications of some of those requirements.

The term “real time” has become a buzz word, as in “real time stock quotes”. If the NYSE ticker is running an hour late, your quotes aren't in real time and that's that. Real time, according to Koester, means getting the job done on time, every time. Not some of the time, or most of the time, but every time. This means your primary concern is not average response time, but worst-case response time.

Take a simple example: a processor that controls a servo based on one analog-to-digital converter's input. The processor reads the A/D converter and adjusts

the servo accordingly. Painting pretty pictures based on the data or shuffling it off to a log file are secondary.

Linux, as Linus Torvalds and his team of kernel hackers provide it, simply is not a real-time operating system, nor was it ever intended to be one. It is designed to provide the best average response time. As Linux is loaded down, it gracefully degrades the performance of all tasks. This is not acceptable for real-time computing. The data acquisition and control functions must respond on time, every time. They cannot degrade, gracefully or otherwise.

The RTLinux response to this is to write a minimal RTOS, then run Linux as a background task under the minimal RTOS. As long as the real-time functions are getting done, and there are resources to spare, Linux is permitted to run whatever tasks are assigned to it. The typical RTLinux application runs one or two real-time tasks. They collect data and stuff it into FIFOs. All processing is done in the background, using Linux and all the neat programming tools that come with it.

Desktop Real Time

This makes RTLinux suitable for what we might call “desktop real time”: data collection where overhead such as several hundred megabytes of Linux are acceptable, even when you don't use them in the application. It means applications where a typical desktop computer, with its keyboard, video display (even running X) and other extraneous processes are acceptable overhead. The extreme here is a student project which collects a few streams of analog data and uses them to control some device.

While RTLinux is excellent for desktop real time, it is by no means restricted to the desktop. Applications using RTLinux running on stripped-down PC hardware can be embedded into products.

RTLinux also offers very fast development for real-time programming. The data collection driver is a Linux module, which means you can remove it, recompile it and re-insert it without rebooting. All of the post-collection data messaging is done with the tried-and-true tools we all know and love, such as Perl and Ghostscript.

Also, RTLinux offers very inexpensive R&D economics. If you have a desktop computer with Linux on it, you already have most of the capital cost of implementing a minimal RTLinux project. You can spend money for custom PC-compatible hardware if volume and real estate are constraints. Many RTLinux projects will do this.

Embedded Real Time

However, Cygnus wasn't after “desktop real time”. The term “embedded” originally meant putting a microprocessor into some product in such a way that it wasn't obvious a computer was even there. A classic case is a photocopier. Even a simple home photocopier has some microcontroller running it. But it doesn't look like a computer: there is no QWERTY keyboard, no monitor, no GUI. The key to an embedded application is that absolutely no inessential overhead is acceptable.

There are two classic embedded project paradigms. The first is a mass production product, where tens of thousands or millions of pieces will be manufactured. Spending hundreds of thousands of dollars up front to shave ten cents off a single unit is economical because of the production run involved. An automobile brake controller is an archetypal application.

The other classic embedded project is where the customer is the European Space Agency (ESA) or the Department of Defense, and spending vast piles of money to shave a gram of weight is considered economical. For example, the Jet Propulsion Lab's Mars Pathfinder or Voyager spacecraft fit this paradigm.

Cygnus decided it was simpler to start from scratch and build an RTOS for embedded applications than it would be to strip unnecessary components from Linux to produce an RTOS suitable for embedded projects. eCos was designed to scale down to just a few kilobytes of code if all you require is a HAL and C library. An attempt to strip Linux that far down would produce something you would not recognize as Linux.

An RTOS suitable for embedding in applications where lives are at stake, such as automobile brakes or pacemakers, must be provably reliable. An application where high production runs are planned must also be highly reliable, because the costs of replacement are much higher than the costs of making sure the product is reliable.

A key component of reliability is simplicity: the simpler an RTOS is, the faster you can prove its reliability because there are fewer interactions to test. So the time-to-market you may gain in fast prototyping with RTLinux, you may lose proving that your product is reliable.

Batteries

If an application is to run on batteries or any other limited-power environment (solar panels, radio-isotope powered generators, etc.), a host of considerations come into play. The obvious one is that the hardware must be able to run with limited power. In general, the more powerful a processor is, the higher the

power consumption. The more overhead imposed by an RTOS, the more powerful the processor must be in order to do its job. That translates into shorter battery life, which translates into annoyed customers and lost sales.

Most battery-powered devices have extreme size and weight constraints. The cell phone makes a good example. If your “full-featured” RTOS has a large ROM or RAM footprint, you will pay for that footprint not only in memory cost, but in battery life as well. Similarly, if your RTOS is not efficient and requires more cycles than another RTOS to do the same job, then even if it meets your real-time constraints, it may still blow your power budget.

Not so obvious are the implications of power failures. For example, if a user has just punched a new phone number into her cell phone and the battery dies, she will be annoyed if the new number wasn't saved. You cannot afford file corruption caused by power failure. This means the designer has a tight window between the time the last digit is pressed and when the data is saved to permanent storage.

Linux, with its lazy writing to non-volatile storage and the multi-user overhead in its file systems, was not deemed acceptable by Cygnus. A battery-powered application requires an RTOS designed from the ground up for speed, efficiency and utter reliability.

Time to Market

One thing that drives almost all real-time projects is time to market (even student projects are driven by time to grades). Cygnus offers two features that will appeal to the embedded processor engineer.

First is ease of configuration. Having learned from the GNU compiler, Cygnus provides compile-time configuration tools that allow the user to select the components to include or exclude. In addition, the components themselves are configurable, and you have full access to the source for each component and can customize it to suit your application.

eCos achieves globally the sort of modularity that Linux achieves with its drivers. Linux was driven to modules by management issues and to minimize interfaces. eCos was driven to greater modularity by the customers' desire for configurability. As it happens, both get portability as a byproduct, and eCos will gain from the decentralized management made possible by its modularity. That decentralization is essential to the success of an open-source product; something RTOS vendors who make it difficult to access their source have missed. (See Linus Torvalds, “The Linux Edge”, also in *Open Sources: Voices from the Open Source Revolution*, O'Reilly, 1999.)

eCos is currently configured with scripts on Linux and by a GUI program for Microsoft Windows.

Figure 1.

Some RTOSes, designed to target the consumer market, do offer the capability of adding an application framework like a Java virtual machine. Again, open source and the bazaar come to the rescue: third-party virtual machines have been ported to eCos, which allow dynamic addition and removal of applications.

The other feature Cygnus offers is its customer support. When time to market is a big factor, fast, expert support is essential and worth the cost. Cygnus' experience supporting gcc will come in handy here.

This is not to say that you must buy Cygnus' support to use eCos. Cygnus offers an e-mail list for eCos users, for which it charges nothing. It also publishes the source on the World Wide Web. If you want to support yourself by looking through the code, you can do that, too.

Conclusion

Koester contends that eCos and RTLinux don't really compete. The market for RTLinux is desktop real-time and other high-end applications where overhead is less of a factor or not a factor at all. The market for Cygnus' eCos is the embedded processor market, where overhead is a major factor.

Both eCos and RTLinux have their places, and, says Koester, can coexist. For many real-time applications, RTLinux is overkill at the expense of power, price, and product performance. eCos can be driven to the higher end of the real-time spectrum, at the cost of losing the standard desktop applications that make RTLinux so attractive.

While very different in the markets they are intended to serve, Linux and eCos are in fact similar in fundamental concept. Each is a monolithic kernel. Both use modularity, and both are portable because both embody some abstraction of the hardware layer.

Ultimately, says Koester, it is consumer requirements which drive RTOS requirements. Consumer requirements are greatly varied, and so there is room for any number of open source RTOSes.

Resources

Charles Curley (ccurley@trib.com) has worked with computers for twenty years, in real-time embedded projects such as power inverter/battery chargers, theatrical lighting controllers and data collection computers. He has written compilers and operating systems for such applications. He has written embedded projects in assembler, C and Forth, and used 8, 16 and 32-bit processors.

Archive Index Issue Table of Contents

Advanced search