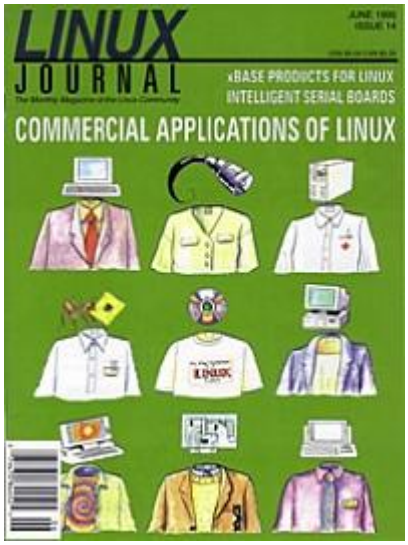
Linux Journal Issue #14/June 1995



Features

Introduction to Eiffel *by Dan Wilder*
All four compilers for the new Eiffel language are available for Linux. Dan Wilder introduces us to the language.

Review: xBase Products for Linux *by Robert Broughton*
Robert Broughton reviews two products, FlagShip and dBMAN, which provide xBase, the de-facto standard database language for PCs, on Linux.

Review: Intelligent Multiport Serial Boards *by Greg Hankins*
Breg Hankins reviews four multi-port serial boards with Linux support.

News & Articles

Linux at the UW Computer Fair
Linux at Comdex/Fall: A Call for Participation *by Mark Bolzern*
Interview with Mark Bolzern
Caldera and Corsair

Reviews

Product Review SlickEdit *by Jeff Bauer*
Book Review Running Linux *by Grant Johnson*

Columns

Letters to the Editor

Introduction to Eiffel

Dan Wilder

Issue #14, June 1995

Dan introduces us to the Eiffel language, a fairly new Object-Oriented language designed to streamline the software engineeering process.

In the late 1970s, when I was a journeyman laboring over the macro assembler language, a simple, elegant, and expressive language called C emerged from the structured programming tradition. My friends advised me to forget C, to learn a “real programming language” like PL/1 or Ada, available on many more of the important platforms, and with commercial staying power due to the support of the government and major corporations. Ignoring this advice, I concentrated on C. Some of you may agree my choice was the right one for its time.

Now, more than a decade later, another simple, elegant, and expressive language has emerged, this time from the object-oriented tradition. With roots in Simula, beholden to no backwards compatibility issues, the Eiffel programming language was designed from scratch by Bertrand Meyer and his colleagues for the purpose of enabling the construction of robust, general, reusable software components. With commercial support from a few vendors, availability on several significant platforms, and some success stories in development of major commercial applications, this language bears careful observation.

I have known of Eiffel since 1989, when I read Dr. Meyer's book, *Object Oriented Software Construction* (Prentice Hall, 1988). I was impressed with the language, much as I had been impressed with C when I read Kernighan and Ritchie's classic, *The C Programming Language* (Prentice Hall, 1988), ten years earlier. The opportunity to give Eiffel a try presented itself with the Linux port announced in 1994 by Bertrand Meyer's company, Interactive Software Engineering of Goleta, California.

The Pitch

So what is all this noise about object-oriented programming, and what sets Eiffel apart from the pack of object-oriented languages?

Much of the promise shown by object-oriented software construction may be attributed to the possibility of code reuse, particularly the reuse of software components.

By reuse, I mean the incorporation of previously written software *unmodified* into new programs. In the US we have a saying: “if it ain't broke, don't fix it.” This folk wisdom highlights the result of research studies which indicate the great hazard of introducing new problems when working on existing software. Ideally, it should be possible to write something once, then put it on ice to be reused but never modified, except to fix bugs and perhaps to add new features. Existing users should be protected, if possible, from effects of such changes.

Unfortunately reuse has succeeded in only limited ways so far. For example, the Unix C library or the various widget libraries used in constructing Graphical User Interfaces are widely used across many platforms. These notwithstanding, most of the significant computer programs produced today contain great volumes of handcrafted code. The difficulty of producing software components with enough built—in flexibility is prominent—and a great part of this difficulty must be ascribed to language issues.

The Eiffel programming language was designed to promote re-use. Bertrand Meyer recounts in the preface to *Eiffel: The Libraries* (Prentice Hall, in press) how he started out to write reusable components and how he came to abandon the attempt to use existing languages and instead wrote a language for the purpose.

Compiled and strongly typed, with genericity (templates), polymorphism, dynamic binding, exceptions, garbage collection, a genuinely useful implementation of multiple inheritance, and unique handling of assertions, Eiffel should be considered a contender on purely technical grounds.

Eiffel provides assertions as language primitives that furnish both in-line design documents and optional runtime error checking. Assertions are inherited. This serves to guarantee that descendents will live up to or exceed their ancestors' promises. Use of these assertions is a part of what is called “design by contract”. These represent an application of responsibility-driven design at a language level.

There are also other factors:

- Eiffel has a published, non-proprietary design, coordinated by a nonprofit consortium whose decisions all existing vendors agree to observe.
- Simple and consistent syntax makes Eiffel an easy language to learn. You will find no dense nests of parentheses, asterisks, brackets, or ampersands in this language. If you delight in special cases and obscure exceptions to rules, with attendant language primitives just for handling these things, Eiffel is not the language for you.
- Precedence of arithmetic operators gives the order of evaluation of mathematical expressions you would expect, unless your native programming language is Smalltalk, Lisp, or Forth.
- Eiffel is available on a wide range of platforms.

The design of Eiffel has been placed in the public domain by Interactive Software Engineering, Bertrand Meyer's company. The Eiffel trademark is owned by NICE, the Nonprofit International Consortium for Eiffel, which has been liberal in bestowing use of the trademark. A validation suite will be available from NICE later this year. Major Eiffel vendors and users, including representatives from corporations and academia, are represented on NICE. Membership is open to any interested party. Proposals of NICE are published on the newsgroup comp.lang.eiffel. Anybody may participate in the ensuing discussion.

The official language definition is *Eiffel: The Language*, By Bertrand Meyer (Prentice Hall, 1992). Almost 600 pages long, this volume contains a precise definition of the language, many examples, and a great deal of discussion. The formal syntax definition occupies only eight pages.

NICE is in the process of standardizing the libraries. PELKS, the Proposed Eiffel Library Kernel Standard, is in the final stages of adoption, even as the vendors hasten to bring their own class libraries into line with it. Other libraries may follow.

Eiffel is available or announced from a number of vendors on a roster of operating systems or platforms including Windows 3.1, VMS, SunOS, Solaris, Ultrix, OSF/1, DG Aviion, IBM RS/6000, Silicon Graphics, Macintosh, OS-2, and NEXTSTEP, as well as Linux.

Many Linux users are interested in seeing vendor interest in our operating system, and a few astute vendors are beginning to join the fold. It seems not too surprising that vendors of a language that is in many respects years ahead of the pack should also be astute enough to recognize the nature of the Linux community. And they have—all four Eiffel vendors offer Linux ports.

Classes

Object-oriented programming draws on just a few main ideas. I will talk about three of the important ones and illustrate their realization in the Eiffel language.

The first important idea is encapsulation: the packaging of data with means to manipulate it. Such a package, written in a programming language is a class, but an instance of a class in execution (or in storage) is an object.

In Eiffel, everything exists within a class. There are no external variables or routines. A class has features. Features in turn are either attributes or routines.

Attributes store values, including references to objects. They may be constant or variable.

Routines do things. Routines are either procedures or functions. Functions return results and are not supposed to change system state. Procedures change system state but return nothing.

All features, even constant or variable attributes, are said to be “called”. This is perhaps less strange than it might seem, for in Eiffel, a call to a function with no arguments is written the same as a call to an attribute. If in some class you write:

```
that := the.other
```

the.other may be either a function or an attribute, in this context it makes no difference.

So, you encapsulate data and the routines that manipulate it in a class. Assertions, mentioned previously, are also parts of a class and serve to express class preconditions, constraints and invariants.

Inheritance

The second important idea is inheritance.

Once you have a class, which describes what you know about the how and why of some sort of object, it may further benefit you to derive a new class from it, with additions and variations, *without* touching the program code in the original class. Inheritance is a mechanism that allows this.

For example, you might derive **BEE** from **INSECT**. Many features of **BEE** would inherit directly from **INSECT**, some features would be modified, and **BEE** would provide a few new features of its own.

A rule of thumb, called the “is-a” rule, furnishes one way to determine whether A might usefully inherit from B. Examine the sentence “A is a B”. Does it make sense? It should if A is a reasonable candidate to inherit from B. For example, “**BEE** is an **INSECT**” passes this test, so **BEE** might inherit from **INSECT**. Then, **INSECT** will be ancestor of **BEE**, and **BEE** will be a descendent **INSECT**.

The “has-a” rule furnishes a contrast. If “A has a B” makes more sense than “A is a B”, it may be prudent to let A reference or have a feature of type B, rather than inheriting from B. “**BEE** has a **STINGER**” makes more sense than “**BEE** is a **STINGER**” or for that matter “**STINGER** is a **BEE**”. Therefore, class **BEE** should have a feature of type **STINGER**. That makes **BEE** a client of **STINGER**, and **STINGER** a supplier to **BEE**.

The client-supplier relationship offers a client less detailed control than a descendent. A client may use or not use a feature of a supplier, but it cannot redefine such a feature. Many of a supplier's features may be hidden from a client, while they will be visible to a descendent. The positive side of this is that the client will be relatively unaffected by details of a supplier's implementation and less likely to be impacted by changes in a supplier.

Both client and inheritance relationships can facilitate software reuse. The traditional function call is more akin to the client relationship, and many attempts at reuse in the past, prior to object-oriented approaches, have made use of the function call. However, we still find ourselves writing and rewriting familiar pieces of functional code too complex to make good candidates for library routines.

The implementation details of inheritance may seriously affect its suitability as a mechanism for reuse. In the ideal implementation, problems arising in descendent classes could always be resolved there. Unfortunately, with many languages, problems arising in descendent classes require changes to ancestors.

This becomes more true with multiple inheritance, a technique by which a class may enjoy, or perhaps not enjoy, multiple ancestors. This technique is a powerful one, but it is unavailable in many object-oriented languages and discouraged in most of the rest. Among languages that have reached commercial viability, Eiffel offers a superior implementation of multiple inheritance.

Polymorphism

In its broad sense, this indicates a situation where a simple request may elicit different but not entirely inconsistent responses, depending on the target of the request. These responses may be arrived at by entirely different means.

For example, you might have classes that look in part like this:

```
class INSECT
-- Description of a standard            -- insect.
     ...
feature flee is
     do
     -- How a standard
     -- insect flees.
     ...
     end; -- flee
end

class BEE
inherit INSECT
     redefine flee
     end;
     ...
feature flee is
     do
     -- How a bee flees.
            ...
     end; --flee
end
class COCKROACH
inherit INSECT
     redefine flee
     end;
     ...
feature flee is
     do
     -- How a cockroach flees.
     ...
     end; -- flee
end

class WATERBUG
inherit INSECT
     redefine flee
     end;
     ...
feature flee is
     -- How a water bug flees.
     ...
     end; -- flee
end
```

Then, you might have examples of **BEE**, **COCKROACH**, and **WATERBUG** bound to references of **INSECT**:

```
-- Define references to an
-- INSECT and to a BEE.

     insect:INSECT;
     bob:BEE;

-- Bind some particular
-- insect to the reference

     insect := bugs.get
```

```
-- Now you have a BEE, a
-- COCKROACH or a WATERBUG.
-- Make it flee after its own
-- fashion, be it that of BEE,
-- COCKROACH, or WATERBUG.
-- This is a polymorphic call,
-- as the code executed will
-- depend on the type of the
-- object bound to insect.

     insect.flee;

-- You can't make a WATERBUG
-- collect pollen.
-- For this you need a BEE, and
-- a trial assignment is
-- available to assign objects
-- that might conform to the
-- type of a reference.

     bob ?= insect;

-- Then maybe you can make bob
-- the bee collect pollen.
-- If he isn't a BEE or a
-- conforming type, bob is Void.

     if bob /= Void then
        bob.collect_pollen
     end;
```

Another sort of polymorphism, sometimes called parametric polymorphism, is supported in Eiffel as genericity.

The final sort of polymorphism I'll mention is found as function overloading in other languages. Here, a function may be defined multiple times, with different types and numbers of arguments. When the function is called, the actual function invoked depends on the argument list.

Function overloading is not implemented in Eiffel. Workarounds are present, and arithmetic operations are handled as special cases, but the general case of function overloading is felt by the designers of Eiffel to be too full of potential ambiguities, type-checking failures, complications, and interactions to be worthwhile just now. A lively thread on this topic is seen from time to time on the newsgroup comp.lang.eiffel.

Multiple Inheritance

Multiple inheritance is a big win under Eiffel. If you have dealt with this in other languages you may be surprised. Multiple inheritance entails many sticky problems, including name collisions and complications of repeat inheritance, and in most other languages is best avoided whenever possible.

When two classes inherited by common descendent have different features of the same name, a name collision obtains.

When a feature is inherited more than once from a common ancestor along two or more inheritance paths, you have repeat inheritance. This may cause a

name collision and also raises practical problems, such as which repeated feature to use in a polymorphic call.

Most object-oriented languages do not attempt multiple-inheritance. The literature is full of elaborate explanations why. This is sad. Eiffel demonstrates that multiple inheritance need not be difficult or complex, and it can also yield some quite practical results.

Eiffel provides an implementation of multiple inheritance which minimizes the adverse effects of name collisions and repeat inheritance complications. As is typical in Eiffel, the solution to one problem helps solve another problem. In Eiffel, a name inherited from an ancestor may be revised in a descendent using a rename clause. Eliminating name collisions then merely involves giving a new name to one or both of the colliding features. The feature is unaffected, except that it is known in the renaming class and any descendents of that class by its new name.

Supposing we have a class named **SOME_OTHER** that inherits two entirely different features both called put from two ancestors named **FIRST_CLASS** and **BOWSER**:

```
class SOME_OTHER inherit
   FIRST_CLASS
     rename
       put as first_put
       end;
   BOWSER
     rename
       put as bowser_put
       end;
     ...
end
```

then in some client class we may have feature **what_now:SOME_OTHER**

```
-- We may invoke the feature
-- named put from either of its
-- originating classes

     what_now.first__put(this);
     what_now.bowser_put(that);
```

Repeat inheritance is only slightly more complex. In the simple case, repeat inheritance is just a by-product of the class inheritance structure you have chosen. Perhaps some of the classes inherited from the class library have common ancestors—this is almost certain. Your responsibility is easy: do nothing. The compiler eliminates the duplicates, and your class has only a single copy of each inherited feature, regardless of how many different ways a feature might come to be present.

In the less common situation, you might want two copies of a feature, for instance put, from an ancestor. A class fragment exhibiting this situation might look something like this:

```
class SOME_OTHER
     inherit
     FIRST_CLASS
       rename
          put as first_copy
       select
        -- Resolve an ambiguity.
         first_copy
       end;
     FIRST_CLASS
       rename
         put as second_copy
       end;
```

The select clause serves to resolve an ambiguity. Suppose you reference an object of type **FIRST_CLASS** and you happen to invoke a feature known in **FIRST_CLASS** as put.

SOME_OTHER inherits **FIRST_CLASS** twice, and because of the renaming, there are two possible answers to the question, "What is the name of put in **SOME_OTHER**?" The following code fragment illustrates:

```
-- Declare a reference of type FIRST_CLASS
-- then attach an object of type SOME_OTHER,
-- a descendent of type FIRST_CLASS, using a
-- creation procedure of SOME_OTHER called "make"

     this:FIRST_CLASS;
     !SOME_OTHER!this.make;

-- When you try to invoke put,
-- without select it is not clear what you mean.
-- Could mean first_copy, could mean second_copy.

     this.put;
```

The select clause removes the ambiguity by saying, "When invoking this duplicated feature under its ancestral name, select this copy." Note that Eiffel brings a thorny problem, repeat inheritance, to the resolution you might hope for. Duplicates are simply eliminated, with no further intervention. If you wish to duplicate-a much less common situation—you can do this using quite simple syntax. Finally, features of the same name that are distinct may be renamed so that both are available, or the unwanted feature may be undefined. pb In each case, the mechanisms provided are simple and local to the class where the problem arises. No revisions to ancestors are required. This is no accident. A major thrust in the design of this language was to eliminate the occasions for revisions to ancestors. Reuse is served here by localizing any adaptations required by repeat inheritance and name collisions in the class where they are encountered. The ancestors are not broken, so they require no fixing. If they are not fixed, they will not have bugs introduced, and other clients or descendents of the ancestor classes will not be affected by changes, bug-

inducing or otherwise, that might otherwise be required in the absence of suitable means to resolve conflicts in the descendent.

Genericity

Suppose you need a class to manipulate a structured collection of objects—an array of **INSECT**s, a parse tree, a hashed list of sales prospects, or some such thing?

Some object-oriented languages furnish a general approach. You construct a template for, say, a **LINKED_LIST** or an **ARRAY**. You then use this template with some arguments indicating the classes to be used in constructing the particular **LINKED_LIST** or whatever.

In Eiffel, this capability is called genericity, and the templates are generic classes. As usual, this is done in a way that does the job and yet is so simple, it seems effortless.

Suppose you are writing an ant hill. First you need some ants.

```
class ANT
     inherit INSECT
     -- A basic ant.  It has features to crawl,
     -- forage, dig, tend the young, and so on.
   ...
end

class CARPENTER_ANT
     inherit ANT
       redefine
       -- Redefine some things.
       -- These chew on your house and your apple
       -- tree.
       ...
       end;
     ...
end

class ARMY_ANT
     inherit ANT
       redefine
       -- These are always on the go.
       -- You hope they don't stop by your place
       -- for dinner.
       ...
       end;
     ...
end
```

Now you are ready for ant hills. Let us suppose you already have some class that models insect societies. Its header might look like this

```
class INSECT_SOCIETY[G->INSECT]
  ...
```

which indicates that an **INSECT_SOCIETY** may be formed using a parameter that conforms to **INSECT**. Loosely, this means any descendent **INSECT** will do.

anthill:INSECT_SOCIETY[ANT] declares a reference to an **INSECT_SOCIETY** containing ants. This reference may then be attached an **INSECT_SOCIETY** containing **CARPENTER_ANT**, **ARMY_ANT**, or any other descendent of **ANT** we have defined. In fact, this allows us to reference an anthill comprising more than one kind of ant, which is convenient, as some anthills may contain more than one kind of ant.

In writing a specific container class, for example, we may wish to take advantage of things we know about insects in the features of the container class. It would never do, in such a situation, for example, to enter an object of class **DOG** or **HAMMER** into this container. The type-safe mechanism for doing this is called constrained genericity and is illustrated above in the header line for class **INSECT_SOCIETY**.

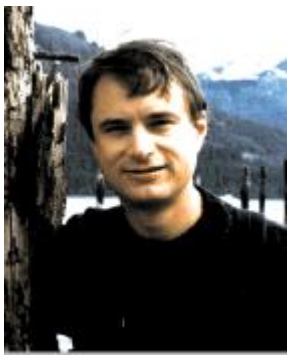
Summary

The Eiffel programming language offers power, simplicity, strong type checking, and numerous other amenities. With an open specification for both the language and the kernel libraries, and support from multiple vendors, Eiffel now stands poised to take off.

According to one vendor, most interest lately has come from people who are turning to Eiffel having used C++ for some years and who have become convinced that the training costs and the complexity of that language are not justified by the features provided.

The more adventurous among us who have a thirst to tackle an object-oriented programming language unhindered by excess baggage from the past ways of doing things may wish to further explore this language.

In my next article, I'll write more about ISE Eiffel and the compiler and tools from Tower Technology of Austin, Texas. I'll also offer a few thoughts about how to get started with this language.



Dan Wilder has been employed as a software engineer since 1975. Dan resides in Seattle, where he divides his time between work, family, and a home Linux

system. Any time left is spent ignoring the moss collection his neighbors think he calls a lawn. He can be reached via e-mail as dan@gasboy.com.

Archive Index Issue Table of Contents

Advanced search

Review: xBASE Products for Linux

Robert Broughton

Issue #14, June 1995

Some Linux aficionados would lke to run their DOS-based xBASE database under Linux. Robert Broughton examines two products that make this possible.

XBASE is a generic term for implementations of what was originally the dBASE programming language. The principal players in the MS-DOS portion of this market are FoxPro (Microsoft), dBASE V (Borland), and Clipper (Computer Associates). It is a language which has statements normally found in programming languages, like IF, ELSE, ENDIF, and WHILE. (There is a GOTO statement, but it is for going to a specific record in the file; There is no GOTO in the FORTRAN flow-control sense.) It also has some powerful statements for processing files and getting data from screens. Setting up relations between files is easy to do. The names of all fields in a file, and their types and lengths, are recorded in the file header. New fields can be added to a file without changing programs which use the file.

None of these products are available for Linux, although there has been at least one effort to make the SCO version of dBASE V work with iBCS. What we have instead is **FlagShip** from multisoft Datentechnik GmbH and **dBMAN** from Versasoft Corporation. Both of these products run on several implementations of Unix; dBMAN also runs on MS-DOS.

These two products resemble each other only at the very simple level. To start with, FlagShip is patterned after Clipper version 5, while dBMAN resembles dBASE III+ or FoxPlus, with some enhancements. This, in turn, means two things. FlagShip, like Clipper, is a compiler, while dBMAN is primarily an interpreter, although it is possible to “compile” dBMAN programs (though not into binaries). FlagShip is also an object-oriented language, which makes it philosophically different from dBMAN, as well as from FoxPro and dBASE. Clipper and FlagShip have a lot of C-like features, which I suspect won't be a problem for most Linux users.

Their target markets are also different. dBMAN is targeted primarily at individual users. If you want a program you can run on your desk to keep track of time billed to clients, or that will maintain a data base of customers or your inventory, dBMAN will do the job, but FlagShip might be overkill. FlagShip is aimed at people who want to develop or port software packages.

I consider four features to be essential for developing decent user interfaces: pull-down menus, windowing, hot-key activated selection lists, and input validation. These products were evaluated with these criteria in mind. I'm also in awe of FoxPro's BROWSE command, so it's the standard by which I measure the browse features of other languages.

FlagShip

As I mentioned previously, FlagShip is a compiler. It translates the xBASE code into C, and the result is turned over to the gcc compiler. (Yes, you can link functions written in C or assembler into FlagShip programs. You can even mix xBASE and C code in a program.) The compiled binaries may be distributed without paying royalties to multisoft.

FlagShip has no equivalent of the “dot prompt” or interactive command interface found in other xBASE products. However, there is a public-domain program in WorkGroup Solutions' FTP area named **dbu** . This program will provide the capability to create files and indexes, add, change, or locate records, and browse files.

Figure 1

The FlagShip feature I like best is its on-line reference program called **fsman**. **fsman** contains the entire FlagShip manual, over 1,000 pages worth of material. If you use an X-window environment, you can easily park fsman in one corner of your screen, and refer to it whenever necessary. It's also possible to save pages of the manual to ASCII files, which makes it easy to incorporate programming examples in the manual into whatever program you are working on at the time. Of course, you could also use the mouse to copy text from fsman into your program.

When you install FlagShip, you get a set of terminfo files specifically for FlagShip. Because of a problem with ncurses 1.8.5, they are compiled with ncurses 1.8.1.

FlagShip doesn't have a function specifically for managing pull-down menus. What FlagShip and Clipper programmers normally do is use **@PROMPT/MENU TO** statements to create the horizontal menu, and use a function called **ACHOICE()** for the vertical menus.

FlagShip has functions for managing windows that work very nicely, but they're not part of the basic package. You have to buy the **FStools** library. As the name suggests, the FStools library is a clone of the Clipper Tools library. There are also windowing functions in the NanForum library, which is public domain.

Hotkeys are set up with the statement **SET KEY keyid TO** statement. Normally, statement would be a function invocation. Within this function, you can call the function **READVAR()** to find out which field the cursor was in when the key was pressed.

An input field can be validated by adding the **VALID** statement parameter to the **@SAY/GET** statement. Again, the statement would normally be a function invocation. Within the function, the value the user typed in could be looked up in a database file.

To determine how compatible FlagShip is with Clipper, I downloaded a couple of programs from a local BBS. I ran into two problems. The programs contained function calls that looked like this:

```
IF (expr, true_result,)
```

FlagShip complained about the absence of the third parameter. Adding ".f." for the third parameter solved the problem. The other problem was a reference to a function named **FT_Shadow()**, which FlagShip simply doesn't know about. I got rid of this problem by commenting it out. Once I got clean compiles on these programs, they worked beautifully.

A key feature of FlagShip is the **TBROWSE()** object. You use this in place of the **BROWSE** command that exists in other xBASE languages. If you don't have any previous experience with object-oriented programming, setting up **TBROWSE()** for the first time is intimidating. However, I was able to get it to do what I wanted by extracting the example in fsman and rearranging it.

I encountered one problem with FlagShip when I was running a FlagShip program in an xterm; I got hieroglyphics instead of line drawing characters. Fiddling with the "**acsc**" parameter in the "fslinxterm" terminfo entry had no effect whatsoever. I worked around this by using the "vga" font that comes with DOSEMU. I eventually learned that instead of using the **acsc** parameter, FlagShip uses another file named Fschrmap.def which maps the character codes generated by the program to the character codes displayed on the screen. I didn't bother to alter this, however. Using the "vga" font is actually a better solution, because it's possible to display double line drawing characters.

Another “gotcha” in FlagShip turned up when I put a **REPORT EDIT** statement in a program; The compiler rejected it. It seems that this statement just doesn't exist in the Linux version. **REPORT FORM** will work if you have an MS-DOS version of Clipper around to set up a report. **REPORT EDIT** does exist in other implementations of FlagShip, and I've been told by WorkGroup Solutions (the North American agent for FlagShip) that it will be in the next Linux release.

Another problem had a number of people scratching their heads for a while. **ACHOICE()** and several other functions were simply refusing to work. It turned out that the name of my program was browse.prg, and **browse** is treated as a reserved word by FlagShip. I can report that the support people at multisoft and WorkGroup Solutions were reasonably helpful in resolving this problem.

WorkGroup Solutions is very aggressive at marketing their product. (They also sell a Linux CD-ROM.) They turn up quite frequently in the comp.language.xbase and comp.os.linux.misc newsgroups. They promise that FlagShip will eventually have some sort of GUI support. Programs that have been ported, or are in the process of being ported to FlagShip include:

- **Accounting for Clipper**--A demo version can be found in WorkGroup Solutions' FTP site, but it's considered alpha
- **SBT**--A MAJOR accounting package
- **SourceMate**--Another accounting package
- **Comix**--A library to handle FoxPro-style compound indexes
- **BandIt**--A report writer
- **SoftCode**--A program generator
- **xPRINT**--A printer control library
- **NanForum Kit**--A widely used function library

WorkGroup Solutions also happily points out that FlagShip programs could be attached to WWW pages, making it possible for net surfers to access and update databases.

dBMAN

dBMAN functions primarily in interpretive mode, although it is possible to compile programs. (Compiling a program does not produce an executable binary; It produces a .run file, which still requires dBMAN to execute it.) When dBMAN starts up, a **CMD:** prompt appears at the top of the screen. At this point, you can type in **ASSIST**, which starts up a menu-driven interface similar to ones available with FoxPro or dBASE, but limited in comparison; dBMAN's **ASSIST** only allows one file to be open at a time, which in turn means that it is not possible to set up relations. It is possible to start up a simple program generator from **ASSIST**. Again, it has a single file limitation.

It is also possible to enter **CREATE REPORT** or **MODIFY REPORT** at the **CMD:** prompt. This will put you in dBMAN's report writer, which works very nicely. The report writer allows relations. I had a little bit of trouble getting it to write a report with lines wider than 80 columns, but I eventually got it to work.

dBMAN provides a function called **PMENU()** to create pull-down menus. **PMENU** doesn't have any mechanism for temporarily disabling a menu choice.

dBMAN handles windows differently from other xBASE products. Prior to defining a window, you call **PUSHWIND()** to push the current window onto a stack. (When a program is in its initial state, the entire screen is considered to be a window.) You then call **WINDOW()** to create the window. When you are finished with it, you call **POPWIND()**, which removes the window, and makes the previous window active.

dBMAN allows you to define only one hot key. You do so by invoking the ONKEY( ) function. This will have no effect until you execute the statement **ON KEY *statement***. (***statement*** will normally be **DO hot-key-handler**.)

The **BROWSE** Command has a long list of options. You can browse only certain fields, and you can specify the width of each field, and whether it is editable. The list of fields can include fields in other files, which is great if you have relationships set up.

dBMAN does not use either termcap or terminfo. Instead, it includes a file named dbmterm.dbm. It looks a lot like termcap. The first problem I had to solve after installing dBMAN was that there were no entries for either "**xterm**" or "**console**".

I created one for color xterms without a whole lot of difficulty, and it is included in Listing 1.

dBMAN has no facility for executing functions written in C or assembler.

There were a couple of nasty bugs in the version of dBMAN I evaluated, which was 5.32. The main one was that procedure files simply didn't work if the procedure file was a .prg. If you compiled it into a .run file, it worked OK.

A Benchmark

I put together a simple benchmark program, which can be found in Listing 2.

The test file I used contained 33,830 records. I ran the benchmark with dBMAN (compiled and non-compiled), FlagShip, and FoxPro 2.0 under MS-DOS. The benchmark was done on a 66MHz 486 with a SCSI disk. Here are the results. At

first glance, you might conclude that both dBMAN and FlagShip were blown out of the water by FoxPro. This would be unfair. FoxPro generally beats similar MS-DOS products in benchmarks, because FoxPro, by design, grabs all resources it can find. No well-behaved Linux program would do this. To put it another way, dBMAN and FlagShip would run a lot faster if they allocated most of the 16 megabytes of memory on my machine, but someone doing text editing on another terminal would see their performance suffer.

Compatibility

xBASE files always have separate data (.dbf) and index files. The format of data files is pretty much uniform for all xBASEs, but as far as I know, no two xBASE products use the same index file formats. I was able to use the same .dbf files with FlagShip and dBMAN, but I haven't tried any memo fields yet. (Memo fields put free-form text into a separate file, usually with the .dbt extension.)

Resources

Robert Broughton (a1040@mindlink.bc.ca) has been developing software for 23 years, and has been using Linux since February, 1993. He is employed by Zadall Systems Group, in Burnaby, BC, Canada.

Archive Index Issue Table of Contents

Advanced search

Review: Intelligent Multiport Serial Boards

Greg Hankins

Issue #14, June 1995

Comtrol, Cyclades, DigiBoard, and Stallion boards are tested in *Linux Journal*'s hardware review. If you need more serial ports than you currently have, read on.

If you are planning on supporting a cluster of terminals, a modem pool for a BBS, a SLIP/PPP server, or a UUCP site, you might find yourself quickly running out of serial ports. Standard PC serial ports are limited to four ports, so chances are if you need more than four ports you may be considering buying multiport boards. There are many factors to consider when adding multiport serial boards to your machine. As modems get faster and faster, I/O throughput has become an increasingly important factor. It's also important to consider CPU usage when adding a large number of ports. As you add more ports, the host CPU will have to spend more time doing serial I/O.

“Baud vs. bps”

There are two basic types of serial boards: “UART-based” (Universal Asyncronous Receiver Transmitter) and “intelligent”. Standard PC serial boards (COM1-COM4) typically come with 8250 or 16450 UARTs with a one-byte transmit and receive FIFO (buffer), or 16550A UARTs with sixteen-byte transmit and receive FIFOs. Boards such as the Boca 2016 and the AST Fourport use these types of UARTs in a multiport board configuration.

Most of these types of boards are supported by the standard Linux serial driver, since they all use the same types of UART and I/O technique. Due to limited FIFO size, and the fact that all character processing must be done by the host CPU, a UART-based serial board might not be sufficient to provide the I/O power you need, for example, to drive a high-speed modem bank.

Table 1. Supported Multiport Serial Cards

This is where the intelligent multiport serial boards are useful. These boards have serial port controllers with larger FIFOs and with some sort of "intelligence", such as RISC UARTs with some character recognition and flow control logic. Some may even have a CPU of their own to handle serial I/O. Since these boards vary in chipsets and control logic, a driver must be written for each board for use with Linux. With the old tty driver (which provides the abstract, general handling for all terminal devices, including serial, console, and pseudo-tty's), support for intelligent multiport boards was nearly impossible.

In the 1.1 Linux development kernel, Ted Ts'o (the maintainer of the serial driver and the tty driver) rewrote the generic tty driver to allow support for many kinds of serial devices, improving the serial drivers in the process. Since then, several drivers for intelligent multiport boards have been written, and more are being developed. In this issue, I'll be reviewing 8-port intelligent serial boards supplied by four manufacturers: Comtrol, Cyclades, Digi International, and Stallion Technologies. For vendor contact information, please see the Vendor Contact Information. Now let's look at each board in detail.

Hardware Features

The following 8-port serial boards were sent to me for review. It should be noted that many other models of the boards, with different port configurations and I/O capabilities, are supported under Linux in addition to the ones I reviewed. A list of supported boards can be found in table 1. In the Control Signals column of table 1, MC stands for "*RS232 modem control*" and HWC stands for full "*RS232 hardware flow control*". In the Max speed column, all figures are in bps.

Comtrol RocketPort RA 8

The RocketPort RA (Remote Access) 8 series of boards features two 36 MHz ASICs (Application Specific Integrated Circuits) with 256-byte transmit FIFOs and 1024-byte receive FIFOs for each port, and built in flow control and line discipline handling. The ASICs also handle other functions, such as the bus interface logic and other miscellaneous logic, significantly reducing the number of components on the board.

RocketPort RA 8 boards support full RS232D modem and hardware flow control signals at speeds up to 230.4 Kbps. Four RocketPort RA 8 boards can be installed in one system. The first requires a 68 byte I/O address range, and each additional board requires a 64 byte I/O address range. I/O address ranges are selectable from 0x100, 0x140, 0x180, 0x200, 0x240, 0x280, 0x300, 0x340, 0x380 and are set with a DIP switch. No IRQ is needed for any of the boards. The driver comes in the form of a loadable module and is supported by Comtrol. The RocketPort RA 8 series features the RocketPort RA Octacable, which

includes an octopus cable with DB25 connectors, at US $499.00, and the RocketPort 8 RA, which includes a connector box with DB25 connectors, is priced at US $678.00.

Cyclades Cyclom-8Y

The Cyclom-8Y series of boards features two 12.5 MHz Cirrus Logic CD-1400 RISC UARTs. The CD-1400 UARTs handle flow control and special character recognition and also have 12 byte transmit and receive FIFOs, as well as a holding and shift register for each port. The Cyclom-8Y boards support full RS232C modem control and hardware flow control signals (except for the Cyclom-8Ys, which have no RTS signal) and will support speeds up to 115.2 Kbps. Each Cyclom-8Y board needs one interrupt (IRQ) selectable from 5, 9, 10, 11, 12, and 15, and an 8K block of dual-ported RAM selectable from 0xA0000-0xEE000. Both IRQ and I/O address are set with a DIP switch on the board. Four boards can be used simultaneously, each requiring its own IRQ and I/O address.

The driver for the Cyclom boards is included in kernel sources starting with version 1.1.74 and newer. The Cyclom-8Y boards are supported by Cyclades, which was the first company to offer a vendor-supported driver for an intelligent multiport board under Linux. There are four 8-port models in the Cyclom-8Y line: the Cyclom-8Ys with on-board RJ12 connectors (no RTS signal), at US $459.00; Cyclom-8Yo with an octopus cable with DB25 connectors, priced at US $511.00; and the Cyclom-8Yb and Cyclom-8Yb+ with DB25 connectors in an external box (the Cyclom-8Yb+ also has surge protection), at US $599.00 and US $699.00, respectively.

DigiBoard PC/8e

The DigiBoard PC/8e board is driven by an 12.5 MHz Intel 80186 CPU to handle I/O processing. It also uses 64K on-board RAM for data buffering. Seven PC/8e boards can be used in one system, each requiring one four-byte I/O address selectable from 0x100, 0x110, 0x120, 0x200, 0x220, 0x300, and 0x320 with a DIP switch. The nice thing about this board is that the DIP switch is located on the back of the card, so you can see and change the I/O address without opening your computer. An 8K block of dual-ported RAM is also required, but this can be shared among all boards. This address is selectable from 0xC0000-0xEFFFF by the PC/8e driver. No IRQ is required.

PC/8e boards support RS232C full modem and hardware flow control signals at speeds up to 115.2 Kbps. The driver for this board is supported by Troy De Jongh (an employee of DigiBoard), not by DigiBoard. The PC/8e host adaptor is US $795.00 plus US $90.00 for a DB25 connector octopus cable, US $110.00 for a DB25 connector box, or US $110.00 for an RJ45 connector box.

Stallion EasyIO/8, and EasyConnection 8/32

Stallion boards employ the same CD-1400 RISC UARTs as the Cyclades boards, with the same features. There are two models of the Stallion 8-port boards: the EasyIO/8, and the EasyConnection 8/32 modular board. Both boards need an 8 byte I/O address, selectable from 0x200-0x3FF with a DIP switch, and an IRQ selectable from 3, 4, 5, 7, 10, 11, 12 and 15, with the driver. The IRQ can be shared by all boards on an EISA bus machine. The EasyConnection 8/32 board needs an additional 32-byte secondary I/O address, selectable from 0x200-0x3FF via the driver, also sharable among all EasyConnection 8/32 boards. Any combination of four EasyIO/8 and EasyConnection 8/32 boards are supported by the driver.

EasyIO/8 boards offer 8 RS232C serial ports at speeds up to 115.2 Kbps, with full modem and hardware flow control. EasyConnection 8/32 boards offer 8-32 ports supporting RS232D full modem and hardware flow control signals at speeds up to 115.2 Kbps and, optionally, RS422A signals, also at 115.2 Kbps. The EasyConnection 8/32 is offered with 8 or 16 port modules, which can be used in any combination together to form a 32 port module. Each module has its own CD-1400 chips to support the ports on that module. The modules are nicely made and also have wall mount brackets on the back. The driver for this series of boards is supported by Greg Ungerer (an employee of Stallion), not by Stallion. The EasyIO/8 is available with a DB25 connector octopus cable, or an RJ45 connector box for US $595.00. The EasyConnection 8/32 host adaptor is US $300.00. Eight-port modules are priced at US $595 for RS232 connector boxes with RJ45 or DB25 connectors, and US $795.00 for a RS232/RS422 DB25 connector box. Sixteen-port modules with RJ45 connectors cost US $845.00, and 16-port modules with DB25 connectors are US $945.00.

Common Features

Several common features were shared among all products:

- All manufacturers offers five-year warranties and 30-day money-back guarantees.
- The ISA/EISA buses are the only bus types supported, although support for PCI cards is in progress by some vendors.
- Everyone was extremely helpful; even the people who support the driver in their spare time were very responsive.

Usability Testing

All boards were installed and worked flawlessly following the documentation that was included with the drivers. Each board was used in my system for over a week, supporting my UUCP feed. I also did testing with interactive login

sessions, file transfers, and dialup PPP connections with my V.34 modem. No problems were encountered with any of the boards during this usability testing phase. Dumb terminals were simulated by interconnecting cables on the serial boards. Again, no problems were encountered.

Performance Benchmarks

Unfortunately, benchmarking is a necessary evil for hardware reviews. You just can't judge hardware by its looks, no matter how pretty it is.

Benchmarking is somewhat of a black art. It is possible to tweak benchmarks to produce very biased test results to highlight particular features of a product. I have no connection to any vendors, so my tests are not biased by personal or professional concerns. Also, some benchmarks (such as the ones I did) don't exactly portray real-world situations, but they do provide some sort of performance measurement.

Table 2. Performance

Because of resource limits, I was simply unable to acquire the massive amount of equipment needed to accurately simulate, for example, 8 dialup PPP connections. This would have required 9 computers, 16 high speed modems, 8 phone lines (or another way of connecting the modems), and^well, you get the idea. So, keep all this in mind while reading these benchmarks, and take them with a grain or two of salt.

The most interesting statistics are:

- **I/O throughput**--how many characters are sent and received;
- **CPU overhead**--how much of the host CPU is consumed doing the I/O. Only system time is counted, not user time, because it is the efficiency of the kernel driver and hardware that is being measured.

The ideal board gives the highest throughput with the lowest CPU usage.

Testing Platform

All tests were done on a generic PC, with an Intel 486DX33 CPU and 256K cache, 16MB RAM, and an ISA bus, running Linux 1.2.0. Benchmark tests were done in single user mode, with a minimally configured kernel, to ensure that other program activity would not skew the test results.

Benchmarking Software

The software I used is called **tbench**. It was developed by engineers at DigiBoard, with enhancements made by engineers at Stallion. I consider the benchmarking software to be reasonably unbiased, due to the fact that it was developed by two competitors, and the fact that it is used by yet other competitors (such as Comtrol) indicates that they concur. Further modifications were made by Stallion engineers for Linux, to adapt the software to use **setserial** in order to use higher speeds with the serial ports. The tbench software is in the public domain.

You can get the version of tbench I used at ftp://ftp.cc.gatech.edu/pub/people/gregh/review, along with the full test results. The original version of tbench is available at ftp://ftp.digibd.com/pub/tbench.

Output Tests

The tbench output tests write data to combinations of ports ranging from one port to all 8 ports. The data is written to the output port set as fast as possible, without flow control, to provide a steady stream of data. The data consists of six-digit numbers with checksums. 100K of data is written to each port. Each output test was run three times, and the results were averaged to eliminate any inconsistencies.

There are two sets of test results: “raw” (**-opost**) and “cooked” (**opost**) I/O results. It's important to distinguish which types of activity uses which type of I/O mode. Interactive login sessions use cooked mode for I/O, while programs such as file transfer programs, SLIP, PPP, and UUCP do raw I/O. Cooked I/O is slower, because each character must be examined to see if it's a special character, such as **^C** or **^Z**. In addition, some editing of the line is done. Of course, this takes more CPU overhead. In raw I/O mode, there is no need to examine each character, because all 8-bit combinations are considered to be valid data, and no characters are specially processed.

Output Test Results

Under ideal conditions, the actual character per second (CPS) output will be the serial port speed divided by ten. Each character transmitted is 8 bits plus a start and stop bit, hence we device the speed by ten. Output tests were done at 9600, 38400, 57600 and 115200 bps, each in both raw and cooked mode. The raw output data was compiled into graphs for each speed, showing the CPS throughput on one to 8 ports and the CPU usage on one to 8 ports (see page 50).

Comtrol RocketPort RA 8

This board truly lives up to its name. The RocketPort gave a very solid performance across all ports and at all speeds, even at 115.2 Kbps. CPU usage in raw and cooked mode were the lowest of all boards, except at 115.2 Kbps (where the throughput was still the highest). Throughput ranked near the top at all speeds, sometimes getting slightly less than the Stallion board, and was the absolute best by a margin of 500-1100 CPS at 115.2 Kbps.

Cyclades Cyclom-8Yo

The Cyclom-8Yo gave a somewhat lower performance than the equally-equipped Stallion board. The throughput was nearly always lower, and the CPU usage nearly always higher, compared to the competitors, except at 115.2 Kbps in cooked mode, where the Cyclom bested all boards in CPU usage (but not in throughput) at a surprisingly low 44%. A new version of the driver has been released, in which the CPU handling is improved, but we were unable to test it in time for publication. Throughput also varied irregularly with different port configurations.

DigiBoard PC/8e

The performance of this board was poor at high speeds. CPS throughput was acceptable at 9600 bps, and at 38400 bps in raw mode. At 38400 bps in cooked mode, throughput degraded quickly as more ports were tested. However, Digi does state in their sales literature that 38400 bps is the maximum usage rate for an 8 port board (even though you *can* run the rate up to 115.2 Kbps) so the results weren't all that surprising. At all speeds, the CPU usage in cooked mode was unusually high, and at high speeds, the board ground the CPU to a screeching halt. In raw mode, CPU usage was actually quite good, but the throughput degraded to unacceptable levels at speeds greater than 38400 bps.

Stallion EasyIO/8

Because of the design similarity of the EasyIO/8 and EasyConnection 8/32, and the fact that my tests on the EasyIO/8 were very close to similar tests done by the driver's developer on an EasyConnection 8/32, the EasyConnection 8/32 was not benchmarked. It is reasonable to assume that results for both boards will be nearly identical in an 8 port configuration.

Overall, the EasyIO/8 did quite well. Throughput at 9600, 38400 and 57600 bps was comparable to the Comtrol RocketPort, sometimes winning by a slight margin. Even at 115.2 Kbps, the board performed quite well in throughput, although CPU usages were higher than the RocketPort's at all speeds. Compared to the Cyclades board, which uses the same UART technology, the

EasyIO/8 did higher throughput and generally used less CPU time, except at 115.2 Kbps. The driver must be amazingly tuned to get such high throughput with 12 byte FIFOs. CPS throughput did begin to vary slightly at 115.2 Kbps, but remained steady at other speeds.

Input Tests

Input tests are much harder to do than output tests. Output tests only require one computer, and it's not even necessary to connect any cables. To do input tests properly, you need more equipment. Two computers are needed, each equally equipped and each with the same serial board, of course. These two systems must then be cabled together. One system is then designated as the “producer” system and outputs data, while the other system is designated the “consumer” system and inputs data.

Unfortunately, we lacked sufficient resources to do a good job at this, and so the input tests that were run do not give a worthwhile and reliable indication of the cards' capabilities. We do not want to give questionable test results due to less-than-ideal test conditions, so input test results are not reported here. In future reviews, perhaps we will be able to do meaningful input tests; please tell us if you would find this useful.

Conclusion

1) How much performance do you get for your money?

Throughput and CPU usage are important statistics to consider, besides the price. The Comtrol RocketPort RA boards are hard to beat, both in price and performance. They have the lowest price if you buy the octopus cable version of the board and offer the best overall performance. However, using the connector box makes the RocketPort less price-competitive. The Stallion boards are not bad performance-wise, but the EasyIO/8 is slightly more expensive than the Cyclades and Comtrol boards. The EasyConnection 8/32 is also slightly more expensive, but is a modular board, so the higher price is to be expected. Cyclades boards are quite competitive in the price ranges and are consistently lower-priced than the similar Stallion, but their performance is also slightly lower. The DigiBoard PC/8e is expensive, and performed poorly in our tests. We can only recommend using this board if you have already purchased it.

2) Who supports the driver?

And who can you call when it won't work and you've tried everything? If you are worried about having someone that absolutely has to listen to and fix your problem, then Comtrol and Cyclades have what you need. Cyclades has been backing their Linux drivers for their Cyclom line of boards for almost a year

now, and Comtrol is very eager to support their hardware under Linux. In fact, while doing my benchmarking with the Comtrol driver, I discovered a serious performance problem at high speeds. After I contacted the Comtrol engineer with my problem, he immediately began to examine the driver code and confirmed my suspicions. Later that afternoon, he found the driver problem. I had a fixed version of the driver by the next day.

Even though the other engineers and sales personnel at Digi and Stallion were very helpful, there is no substitute for technical support backed by the vendor.

Resoures

General information on setting up serial devices, such as terminals and modems, can be found in the Serial-HOWTO, available at http://sunsite.unc.edu/mdw/

HOWTO/Serial-HOWTO.html on the Web. Plain text and PostScript versions can be found at ftp://sunsite.unc.edu/pub/

Linux/docs/HOWTO and also at the many sunsite mirrors.

In real life, Greg is an aspiring young sysadmin at Georgia Tech's College of Computing. In his spare time, he maintains the Linux Serial-HOWTO and coordinates the HOWTO project. He has been running Linux for over two years now. Comments are welcome via e-mail at greg.hankins@cc.gatech.edu.

Archive Index Issue Table of Contents

Advanced search

Linux at the UW Computer Fair

LJ Staff

Issue #14, June 1995

This year, our booth was one of the most popular booths.

On March 15 and 16, 1995, the University of Washington had its twenty-first annual Computer Fair in Seattle. In 1973 it was called the Terminal Fair because, after all, people couldn't afford computers back then. Now this has grown to an event with around 16,000 attendees ranging from students to Boeing and Microsoft engineers.

SSC, publishers of *Linux Journal*, has had a booth at the fair for the past four years. In 1991 and 1992, SSC's booth with Unix products was not very popular among attendees compared to the other booths with mostly Windows and MS-DOS products. In 1993, we saw increased interest in our booth. Last year, in the Spring of 1994, we gave away several hundred free copies of Issue 1 of *Linux Journal*, we talked to hundreds of interested people about Linux.

This year, our booth was one of the most popular booths. One SSC employee (biased, we admit) claimed that only one booth at the show was more popular than SSC's booth, and that was the Starbucks Espresso booth. (For those not familiar with Seattle's reputation, it prides itself on the number of quality espresso stands and is called “latte-land” by some. Allegedly, Seattle consumes more coffee than any other city in the world, save Milan, Italy.)

This year, we gave out 4000 copies of *Linux Journal* and had hundreds of people enter their business cards in our drawing for an Essential Linux Pack. The winner of the Pack was Don Herold of North Bend, Washington.

Common questions asked while in the booth were:

- How big a machine do I need to run Linux? Most people were surprised to find out machines much smaller than what are needed to run MS-Windows will support Linux just fine.

- Will Linux run my DOS or (MS-Windows) programs? Most people were happy with a “yes” to the first question and seemed to settle for “maybe sometime soon” for the second part.
- What does Linux cost? “Nothing” was a surprising answer for many. Handing them an SSC catalog that includes various CD distributions, as well as instructions on where and how to get Linux off the Internet, seemed to get people thinking that Linux for free is a reality.
- Does Linux run on a Mac? Not a really popular question but when we explained to people that Mac programs can run on Linux (with the addition of Executor) these people seemed very surprised.

In addition to the booth, Phil Hughes, publisher of *Linux Journal*, spoke to a packed auditorium of 500 people. Phil's talk followed an IBM presentation on OS/2 Warp which, while well-attended, was not a full house.

Phil abandoned the more traditional marketing-oriented talk and, instead, presented a Question and Answer talk where he supplied the questions. When asked the reason for this type of talk he responded, “Most people interested in Linux don't need a sales pitch. They just want to know what it is and if it will do what they need done. If it does, it will sell itself.” The large crowd at the *Linux Journal* booth following the talk indicated that Phil was right.

On Monday, March 20, the Seattle Linux Group met at SSC's offices. Prompted by the publicity at the UW Computer Fair, 58 people attended, up from the more typical 15-25 people. One person at the meeting told the group that he had not really been thinking about Linux but after talking to us at the Fair, he purchased a CD and installed it over the weekend. He was very pleased with the results.

Archive Index Issue Table of Contents

Advanced search

Linux at COMDEX/Fall

Mark Bolzern

Issue #14, June 1995

Linux International, *Linux Journal*, and WorkGroup Solutions have arranged for Linux to have a major presence there. But, there is a problem. We need help. This show is bigger than any of us.

COMDEX/Fall is America's largest computer trade show. It usually has 200,000 attendees, and worldwide is second in size only to CeBit in Hanover, Germany. Traditionally, COMDEX is largely for resellers of computer technology, and is the show most often selected by major vendors for important product announcements. Many successful products trace their early launch to COMDEX Fall.

Linux International, *Linux Journal*, and WorkGroup Solutions have arranged for Linux to have a major presence there. But, there is a problem. We need help. This show is bigger than any of us.

We would appreciate it if any Linux enthusiasts going to COMDEX, or living within driving distance of Las Vegas would volunteer their help.

We need 3 things to pull this off:

- Companies to take (and pay for) booths in the Linux Showcase area on the show floor. Contact Richard Kazarian at SoftBank, (617) 449-6600, and tell him that you want to be part of the Linux Showcase. Also, please let me know that you are participating. If you cannot take a booth, then become a corporate sponsor for Linux International, (www.li.org for info) and bring your literature to distribute in LI's booth. Better yet, volunteer to help man LI's booth, and present LI and your product, too.
- Volunteers to submit papers for sessions they would like to present in the conference track that is being planned. Any subject is good, from “My first experience with Linux”, “How my company uses Linux”, to technical

subjects on the design of the kernel, how to configure networking, etc. Virtually any subject is fair game. Please do not be shy about suggesting subjects. The range of people attending will be from people who have never heard of Linux to some of the few experts that actually exist. Your name and session information will be printed in the show conference guide, possibly your picture published, and you can use this as an impressive reference for any future resume, work or presentation you do. Volunteering gets you free admission to all conference sessions and the show floor, if your suggestion is accepted. Unfortunately, we cannot pay for travel expenses for anyone.

You may note that, of all things, OSW (Open Systems World, in Washington D.C.) and COMDEX are the same week. For those people who would like to be present at both shows and end up in Vegas for a party weekend, Phil Hughes, publisher of *Linux Journal*, and I are arranging sessions at OSW the first part of the week, and at COMDEX the second half.

- Volunteers are needed to help in Linux International's booth, and possibly the booths of Linux Vendors who request it. Also, volunteers are needed to hand out literature to draw people to the Linux Showcase area. Volunteering gets you free admission to the show floor where the vendors are.

If this is handled well, this one event could serve to launch Linux into commercial acceptance. Please help us make Linux known as THE OS, and THE Revolution.

- For papers, offers of help, or questions, please write to: Mark Bolzern at shows@wgs.com.
- For information on COMDEX/Fall in general, contact Steve Engle or Richard Kazarian at SoftBank, (617) 449-6600, and tell them how excited you are that COMDEX/Fall will have a substantial Linux Focus, and that this is the reason you will attend (if true).

COMDEX/Fall will be held in Las Vegas, Nevada, USA, November 13-17, and it takes over the entire town. There are lots of parties, free food, give-aways and fun for all. If you have never been at COMDEX (everyone should do it at least once in their lives), be warned—you are likely to run into any of the people whose names you regularly see in print, from Gates to journalists. You may even speak with them. See you there.

Archive Index Issue Table of Contents

Advanced search

Interview with Mark Bolzern

Linux Journal Staff

Issue #14, June 1995

Mark Bolzern talks about a recent trip spent promoting Linux and Flagship, the product he sells for Linux and other Unix-like systems.

Mark Bolzern is the President of WorkGroup Solutions, Inc, a company selling the FlagShip CA-Clipper language product, and is also a board member of Linux International. When we heard that he was taking a tour promoting Linux, we asked him to share that experience with our readers.

LJ: I understand that you just completed a tour to promote Linux and FlagShip. Where did you go?

Mark: I started with the CA-Clipper Users' group in Detroit. I then went to Boston and met with magazine editors out there, including Byte, Information Week, and a few others. I also met with several well-known industry analysts while I was there. My next stop was Cleveland for a presentation to the Greater Cleveland Clipper Users' group. Then I went to Chicago and spent time educating our PR firm on Linux. The last leg included presentations to Clipper users' groups in St. Louis, Jefferson City, and Kansas City. The whole trip took about 3 weeks. I drove my van rather than attempting to fly, because of the equipment I had with me, and because the plane schedules were such that it would have been a lot more hassle to fly.

LJ: Were these normal users' meetings that you attended, or did you do anything special to bring in other users?

Mark: I had my PR firm put on radio spots and local newspaper and magazine ads. These drew additional people in. Mailings were also done.

LJ: How many people did you introduce Linux to on this tour?

Mark: Hundreds of people saw Linux for the first time, were impressed, and then mobbed me with questions. I purposely did not sell any Linux CDs on this tour, but I gave away quite a few as door prizes. In fact, here is a quote from Mark Schumann, President of the Cleveland group. He e-mailed it to me after I left.

> It was nice having you around Cleveland this week. I have to say, Mark, that this is the first GCPCUG Clipper meeting in many months that generated as much excitement among attendees. We've had better attendance at times, and also much worse, but not often do people come and watch and think and mob the speaker as they did this week. Nice going!

LJ: What did your presentation to these groups consist of?

Mark: I had a Pentium 90 running WGS Linux Pro and a big monitor with me. I showed them DOOM, fvwm, Workman playing music from CD, and lots of other things going simultaneously. I also compiled FlagShip programs at the same time and showed off some converted Clipper programs. I waxed eloquent on how Linux is “Unix: The Next Generation”, and why Linux and whatever Microsoft does may be the only operating systems within 5-10 years. I presented a history of the computer industry and the role Unix has played. I also drew analogies to other events in human history that happened for the same reasons that Linux is going to. If you think Linux is something now, “you ain't seen nuthin' yet!” True revolutions have always had to sneak in through the back door. Heck, that is exactly how PCs and PC LANs got to where they are now. The established powers virtually never see the revolution till it is too late. Look what IBM's own problem child, the PC, did to them! From #1 in computers & mainframes to being a “me too” clone company, all because they never really understood it. When I worked for IBM, I asked if I could work with the PC shortly after it came out. I recognized the PC as a revolution in the making. IBM told me the PC would never be a mainstream product. They said if I wanted to work with it, I'd have to transfer to Computerland or IBM's factory in Boca Raton. I quit and went to Computerland.

LJ: What is your “WGS Linux Pro”?

Mark: It is a CD containing the Slackware Linux distribution, along with a number of additional things drawn from the Internet. WGS is specifically targeting the commercial market that present Linux vendors have had difficulty moving into. We provide quality and assurance, not necessarily the latest code. We do not feel that we compete with the InfoMagics, Slackware Pros, and Yggdrasils of the world. We are after people who develop commercial software for commercial use. These are the people who want FlagShip. In the meantime, our PR benefits the entire Linux community. I would wager that all CD-ROM

vendors have sold more CDs since we started our activities, partially due to these very activities. I would like for all CD-ROM vendors to have our FlagShip Test Drives and information on their CDs.

LJ: What is CA-Clipper, and what did you present specifically to its users on this tour? How many Clipper users are there?

Mark: There are over 200,000 CA- Clipper users and programmers in the US alone. If you add the similar Fox and dBase products, there are well over 10 million. Clipper is a very popular applications development language worldwide; in fact, it is even more popular outside the US than inside. Just as Cobol is the legacy language to mainframes, CA-Clipper is the legacy language for business applications on the PC. There are millions of business software packages written in it. But Clipper is far more than just a legacy language, it is still a concept leader. Clipper started as a way to compile dBase III programs and then began to add features and functions. When Clipper 5 was released, it was the first seriously object-oriented language for creating business applications. It is now one of the very best languages for developing character-based data-oriented applications. It allows fast prototyping, simultaneously creating robust and easy to maintain code. The programs created can be distributed completely royalty free and with no runtime module necessary. There is a large community of developers of add-on tools and products for Clipper—as many vendors as probably for any other development tool. There is much more information I could pass on, but people can feel free to check out our FTP site for further information, URL ftp://ftp.wgs.com/pub2/wgs/Filelist, or write info@wgs.com. They could also pose questions on the alt.comp.databases.xbase.clipper Usenet forum or the CompuServe Clipper Forum, or call Computer Associates (CA) for information on Clipper.

LJ: FlagShip is CA-Clipper for Unix?

Mark: Yes, call CA and mention Clipper and Unix in the same sentence. They will send you to WGS. When people mention a DOS version of FlagShip to us, we first try to sell them Linux and, if it doesn't work, we send them to CA for Clipper. So, in summary, FlagShip is an implementation of CA-Clipper for Unix and there is a Linux version of FlagShip at DOS-level pricing. CA-Clipper has been the dominant language for writing and distributing business applications on DOS, so that means Linux and commercial applications usage could merge at light speed! Millions of applications, no muss, no fuss. Lack of a good, inexpensive, royalty-free business applications language and its resulting applications has held the use of Unix back seriously in most businesses. And products like Oracle, Informix, Sybase etc. are far too expensive. [Read the review of FlagShip and dBMAN in this issue of *Linux Journal*—ED]

LJ: Why do you call Linux “Unix: The Next Generation”?

Mark: I explained that a bit in my editorial last month, A new project or a GNU project? You can look at it this way. Unix is very modular, with the basic concepts being standard in, standard out, and standard error. These are the interfaces used to build a modular operating system in good building block, Lego style. So it is possible, even easy, to replace pieces of Unix as needed. The GNU project of the Free Software Foundation has been around a long time. It is mostly made up of enhanced and bug-fixed replacement utilities for Unix. Over the years, a lot of Unix manufacturers have shipped a lot of broken things, and this simultaneously made the GNU project necessary and possible. As a result, when the Linux Kernel came along, most of the Next Generation Unix was already waiting. Voilá, what we know as Linux. Linux is not new; only the kernel, the assemblage and the name are new. Linux is the result of a generation of bug fixes and enhancements by many very astute technical people, not the least of whom is Linus Torvalds. There is another way to look at it, too. The Internet was built primarily on Unix systems and various utilities that were added to Unix such as TCP/IP. Linux was built largely on the Internet, a child of the Internet, if you please. So Unix, and the Internet are parents of Linux —“Unix: The Next Generation”.

LJ: I noticed that you said earlier that you think Linux may be one of only two operating systems within 5-10 years. Why do you think that?

Mark: Hardware manufacturers used to compete with one another by getting the job done better for customers and by trying to create more powerful systems. Then they started doing operating systems software, programming languages, and applications software. Soon language vendors and applications software companies were born because no one manufacturer could do it all. Unix has now been adopted by almost all manufacturers because their own proprietary OS development cannot keep up. The bigger the manufacturer, the longer it took them to come around. The natural next step is for the OS to become totally independent of hardware manufacturers. There are 4 choices: Unix, Linux, OS/2, and whatever Microsoft does. I list Linux separately because it carries the spirit of Unix forward better than any other implementation. Of the options, Linux actually has the most development resources and a cooperative development environment. In addition, all companies and people have equal access to Linux. Microsoft has more money than the Linux people do, but anytime people are hired for pay and position, rather than for the love of it, there end up being many distracting political hassles. Most Linux people are in it because it is art to them. Neither UnixWare nor OS/2 have these advantages, and as such, are behind. POSIX is a committee standard and Linux may become its best implementation in all respects very soon. DEC's interest with Alpha machines is a good example of this. I think that as Linux gets more

capable, stable, user friendly, easy to install, and has applications available, it may actually take over as the primary OS for all hardware manufacturers, specifically because of its independence. It may become the primary target platform for all applications developers that do not want to compete with Microsoft on Microsoft's own turf (OS). Linux should not be called a **free** operating system, it should be called a **priceless** operating system.

LJ: Thank you very much for taking time for this interview.

Mark: My pleasure.

Archive Index Issue Table of Contents

Advanced search

Caldera and Corsair

LJ Staff

Issue #14, June 1995

Who is Caldera, and what is Corsair, really?

Many conflicting rumors about Novell's “Corsair” project have been floating about the Internet. Recently, there have been sensationalized stories that Ray Noorda, formerly the President and Chief Executive of Novell, is backing a company called Caldera to use the “Corsair” technology to take Microsoft on head-to-head. While that makes for an interesting story, at *Linux Journal* we prefer facts to hype. Here is Caldera's story.

Who is Caldera, and what is Corsair, really? Corsair was—and still is—a project at Novell to create a “desktop metaphor” for the network. A year or so ago at Novell, an advanced technology group was doing research on how to better and more easily integrate and manage network access for users, and they decided to focus on the desktop. While they liked Unix, they wanted something smaller and faster to put their “Internet desktop” on, (not to mention something requiring a smaller royalty stream than Unix—yes, Novell pays royalties to other companies that developed parts of Unix).

Several members of the group became convinced that Linux was the best answer to their search. They started to work with Linux, contributing code back to the Linux development team and to other projects related to Linux, including the Linux DOS emulator. Their work included work on the IPX networking layer of Linux, support for the Wine project, and several other smaller parts.

When Robert Frankenberg took over the CEO position at Novell in 1994, he cut out many of the exploratory projects that were then underway in an attempt to focus Novell on “core competencies”. While many in the industry lauded this, it did end all of the work on Linux-related features of Corsair, and Corsair now is an “Internet desktop” for MS Windows.

Several of the members of the group were not satisfied with this, and quit to form their own company, with financial and strategic backing from Ray Noorda, to continue working on this desktop—under Linux. This new company is Caldera.

Myths Debunked

Caldera's product is not an MS Windows clone, as some have reported. It *will* include an MS-Windows-like API, licensed from Willows Software, which is another Noorda-backed company. This will allow companies with MS Windows applications to port those applications easily to Linux. Caldera does *not* intend to have an "ABI" (application *binary* interface) that is guaranteed to run existing Windows applications. Because they have worked for a high level of compatibility at the "API" (application *programming* interface) level, there is a high probability that some MS Windows binaries will run, but they say that is not what they are attempting to accomplish.

Caldera says their product is not intended to be an MS Windows killer: they are not trying to put Bill Gates out of business. They instead wish to provide an alternative: a commercially supported distribution of Linux bundled with commercial components.

Commercial Components

The commercial components, which will require separate licensing, will include the Windows-like API, the Corsair-like desktop, a netware client, and OpenDoc support. Truetype font support is also planned. So-called personal productivity applications will be bundled or sold separately. Caldera-developed documentation will be included as a part of the product.

While Caldera will be providing many commercial components, they have publicly promised to fully honor the GNU Public License, including providing full source code for all the GPL-licensed software they ship. The GPL is what has made Linux useful to them, and they say that it lowers and removes barriers for many small companies who want to compete in the software marketplace. They suggest that Linux will increase innovation in the software marketplace, and they want to push this along. They quote Ray Noorda as saying, "That's exactly what we are out to do—to grow [the whole Linux] industry." Promoting Linux is good for everyone.

Caldera has instructed their public relations firm to promote Linux, as well as Caldera, believing that by giving Linux added exposure, the entire market will grow, benefiting everyone in it, including themselves. In addition, they will continue to contribute work on free software, doing their part to help keep Linux innovative and open. When they chose a business partner to build their

distribution, they chose another company that licenses its software under the GPL, Red Hat Software.

Caldera is not the only company to provide supported, shrink-wrapped distributions of Linux, nor is it the only company to sell commercial applications for Linux. Caldera suggests that they have two distinguishing characteristics: first, they have Ray Noorda behind the company, which gives them credibility and financial flexibility when they are negotiating with large software companies; and second, Caldera will focus on helping and encouraging existing independent software vendors and manufacturers to port their programs to the Caldera desktop in an attempt to provide types of software that have been unavailable for Linux in the past.

Archive Index Issue Table of Contents

Advanced search

SlickEdit

Jeff Bauer

Issue #14, June 1995

Jeff gives us a quick introduction to the SlickEdit editor, a small, efficient programmer's editor which is available on many platforms, including Linux.

MicroEdge, Inc. has released a version of SlickEdit for Linux. SlickEdit is a character-based programmer's editor running on a variety of platforms and it should appeal to developers who work in more than one environment. The current list price is US $195.

For a programmer arriving from a DOS environment, SlickEdit provides a comfortable transition. Instead of arcane keystrokes and endless keyboard remapping, the user is presented with an unobtrusive menu system which may be later discarded. Pressing F1 brings up the help system and the behavior of particular keys, such as insert and delete, act in the manner to which the DOS user is accustomed.

SlickEdit can be considered a fairly complete editor in that it provides most of the features necessary for productive coding. It has full undo and redo capability, even past successive file saves. Multi-file search and replace is supported along with regular expression searches. Cut and paste operations may be performed on marked lines, blocks, and columns. Multiple clipboards of marked text are maintained. A Rexx-like macro language provides over 850 callable editor functions.

A single diskette installs SlickEdit onto /usr/bin and /usr/lib/slick by default, consuming about 2Mb of disk space. Less space is necessary if you choose not to install all the macro files. The SlickEdit executable is only 260K. A 400-page perfect bound manual is included with the application.

There are three ways you may wish to use SlickEdit on a Linux system: console, terminal, or xterm. Each has its own peculiarities and you will probably want to create a separate keyboard description file for each. A key mapping utility

called **genktab** allows you to tell SlickEdit what to expect when a key is pressed. A readable keyboard description file is compiled into a ".tab" file. Another utility, **showkey**, is provided to help you determine what sequences your terminal is generating.

Most of the common terminal types are included in the default keyboard description file, including vt100 and xterm. To work with xterm, the file /usr/lib/slick/xdefault must be appended to the .Xdefaults file in your $HOME directory. A special **xs** script is included to avoid the initial configuration hurdles. Under xterm the window can be dynamically resized with the mouse, but the mouse cannot be used to manipulate text within the screen. MicroEdge has stated they intend to add mouse support for Unix in a future version of SlickEdit.

Under console mode, SlickEdit is virtually indistinguishable from its DOS version. The Alt-F1 thru Alt-F4 keys may need to be remapped to provide functions such as **MOVE-EDGE** and **DELETE-ADJACENT**, since most Linux systems use these key combinations to provide virtual terminals. You also can perform these operations from the SlickEdit menu or configure the Alt key for a non-console terminal as described below.

Setting up a terminal to act the way a normal DOS user expects is challenging, especially for a text editor. The Alt key has no counterpart with most terminals in standard use today. MicroEdge works around this problem by defining the backtick (`) key as the Alt key for a non-console, non-xterm terminal. In practice this adaptation is easy and natural to use. To enter an actual backtick, a **^Q `** sequence is entered.

Another potential problem with using a terminal is common to many Unix applications^-namely the confusion resulting in sending an escape sequence (say, F1 under ANSI emulation) and a literal escape (**0x1b**) character. This is not peculiar to either Linux or SlickEdit, but it is more noticeable in text editing applications where you're likely to be pounding the arrow keys a lot. SlickEdit attempts to alleviate this by allowing the user to specify a delay for an ambiguous key sequence. On other Unix systems I have noticed that this is not always 100% successful, especially if you are running telnet through through heavily loaded systems. However, my limited use on Linux uncovered no problems.

If you purchase SlickEdit, you have, at a minimum, two expectations: ease of use and technical support. After configuration, SlickEdit has very few surprises. Getting to the point that you are satisfied with your configuration, however, may require a phone call to Technical Support. My prior experience with MicroEdge's support has been very good, and this time was no exception. Basically, I wanted true 8-bit line drawing characters rather than the stodgy

hyphen, plus, and vertical bar (**- + |**) symbols displayed. My call was returned in 30 minutes. After ascertaining what I needed, the specialist offered to e-mail me the necessary file, with the assurance that I could call him back if there were any further problems. My ANSI.DAT file arrived later that day and everything worked as expected. MicroEdge also supports a CompuServe forum: “go slickedit”.

My overall impression is that MicroEdge has done an excellent job in porting SlickEdit to Linux. I have written many lines of code over the past two years with the SlickEdit family of editors on a variety of platforms. There are certainly other freeware alternatives to the venerable **vi** and **Emacs** editors, but I find my level of productivity is raised (and frustration lowered) by using SlickEdit.

Resources

MicroEdge, Inc. may be contacted by phone at (800) 934-EDIT or (919) 831-0600, or by fax at (919) 831-0101, or by mail at P.O. Box 18038, Raleigh, NC 27619 USA.

Jeff Bauer has spent the past 16 years developing health care software. His current project involves interfacing pen-based computers with Unix systems to track clinical information.

Archive Index Issue Table of Contents

Advanced search

Running Linux

Grant Johnson

Issue #14, June 1995

Running Linux covers everything you need to install, use and understand the Linux operating system.

- Author: Matt Welsh & Lar Kaufman
- Publisher: O'Reilly & Associates
- ISBN: 1-56592-100-3
- Price: $24.95
- Reviewer: Grant Johnson

There has been an increasing thirst for information about Linux which hasn't been fully quenched by a book...until O'Reilly & Associates got together with Lar Kaufman and Matt Welsh, the Coordinator of the Linux Documentation Project and the author of *Linux Installation & Getting Started*, to take on this project. Unostentatiously titled *Running Linux*, this book is a perfect blend of polished knowledge, organized in an easy-to-grasp package, like most books in the O'Reilly line.

Running Linux covers everything you need to install, use and understand the Linux operating system. This cornucopia includes in-depth installation and configuration instructions, tutorial and discussion of programming tools for system and program development, information on system maintenance, network administration guidelines, and everything in between. The book opens with an explanation of the GNU General Public License and some background concepts. It also includes a brief history of the Linux system; starting with the UNIX operating system that was the inspiration for Linux, then covering its creation by Linus Torvalds in 1991, on to the present day—or pretty close.

Continuing on, you'll find a comprehensive installation tutorial that leads you step-by-step through the tedious task of setting up and configuring Linux on your PC, independent of which distribution you decide to use (although some

examples from the Slackware distribution are given). All the basic concepts relevant to installation are discussed here, and solutions to many common problems are presented.

Next is a presentation of basic UNIX concepts, most of which are not unique to Linux, but which you need to know to take advantage of Linux. You don't want to drive a fancy, powerful operating system without knowing how to shift and steer it. This chapter teaches you how to shift, steer, stop, and even read maps and road signs.

Running Linux also provides complete information on Linux system and network administration. Basic functions such as repairing file systems, installing software, and administering user accounts are made easy. More advanced features such as UUCP, TCP/IP, e-mail, SLIP, PPP, and other serial telecommunications are also discussed. You are even shown how to provide network services from your Linux machine. This includes a section on configuring your very own World Wide Web (WWW) server and writing HTML (Hyper-Text Markup Language) documents for the WWW.

The programming languages and other system tools features in Linux are thoroughly described in both theory and practice. Among these tools are the gcc C and C++ compiler, the gdb debugger, perl, Tcl and the Tk toolkit, the Emacs and vi editors, text formatting systems such as TeX and tools designed to interface with MS-DOS.

Lastly, *Running Linux* offers helpful installation and configuration information to make setting up XFree86 a bit easier. This project alone has been known to cause rashes and other stress-induced ailments, but reading the keep-it-simple approach in *Running Linux* can help keep you healthy...

In short, this book answers the questions the novice users are too afraid to ask, and the questions gurus won't admit they don't know.

Archive Index Issue Table of Contents

Advanced search

Letters to the Editor

Various

Issue #14, June 1995

Readers sound off.

Random Services

I found the article [Setting up Services in the April 1995 issue] interesting. I think that there was one major point that was missed. Administrators should not randomly assign ports to services. In fact there is an RFC that lists the ports. The latest is rfc1700. This is very important for novices to learn.

—Matthew B. Guest mbguest@fastbox.ridgecrest.ca.us

Importance

I agree emphatically with the letter from Graham Leach (g_leach@pavo.concordia.ca) in the April 1995 issue of *Linux Journal*. Your magazine is excellent. But the “linked list” format is annoying. Please avoid it, or at least minimize it.

I disagree with one of your justifications, namely that readers expect “important” articles near the front of the magazine. I certainly don't. I would guess that most of your regular readers, like me, would read each issue cover-to-cover, and more than once. Perhaps you could poll your readers, to find out for sure.

As I said, your magazine is superb. But the format makes it difficult to read, needlessly so.

Regards,

—Carl Renneberg renneber@tmx.mhs.oz.au

Linked Lists again

Just received my first issue of *Linux Journal*: April. Great!

WRT the split article issue raised by Graham Leach in a letter to the editor, may I suggest something that worked well for me back in the dark ages of paste-ups: Instead of "no jumps", which is really tough, how about a goal of no more than one jump per article. It would also be helpful to add the article name to the "continued from..." at the top of jump pages.

Thanks for an interesting, readable book.

—John Miller, N4VU jsm@n4vu.Atl.GA.US

LJ Responds:

We have been discussing this and this issue has a keyword from the article title added to the "continued from" lines, as you suggest. Also, I agree with the one jump max rule and we will try for that. From more information on our new layout, please see Stop the Presses.

Scilab Revisited

I was very pleased to read Robert A. Dalrymple's article on Scilab. I've been using Scilab for 7 years and I think that it is a terrific product (much more powerful than Matlab). It's wonderful that INRIA decided to make this tool available to the public as free software.

My purpose for writing this letter is that I think your readers will be interested in knowing some additional facts about Scilab which did not come out in the article. One of the most powerful aspects of Scilab is its ability to easily perform data abstraction and operator overloading.

Scilab is delivered with several pre-defined abstract data types such as rational polynomials and linear dynamic systems. Overloading of the usual operators such as "+", "-", "*", "/", and "=" allows the user to easily manipulate these abstract data types and the development of higher level operations is greatly simplified. Here is an example with two rationals:

```
^> r1=(2+3*s)/(1+s**2)
   r1        =
      2 + 3s
      ^^^-
           2
      1 + s
 ^> r2=s/(5+s)
   r2        =
        s
      ^^^
```

```
      5 + s
^> r3=r1+r2
  r3        =
                    2   3
      10 + 18s + 3s + s
      ^^^^^^^^^
                  2   3
       5 + s + 5s + s
```

Notice that even though the rationals in the example are user defined objects, the overloading of the “=” operator gives rise to a user friendly on-screen representation which is recognizable as that of a rational. Furthermore since the “+” operator is overloaded for rationals, their sum is defined and requires no special function (like **r3=poly_add(r1,r2)** as an example).

The implementation of the rational abstract data type in Scilab allows all the usual operations that one would expect between two rationals as well as between a rational and other data types (such as scalars and matrices).

The user of Scilab can easily define new abstract data types and develop Scilab macros for the implementation of user transparent operator overloading. These Scilab features are unique in the Matlab-like class of products and is the reason for which I believe that Scilab is a much more powerful product than Matlab.

I hope that these comments will be of use to your readers.

Sincerely,

—Carey Bunks bunks@rechser.total.fr

CORRECTIONS

Archive Index Issue Table of Contents

Advanced search

Changes To *Linux Journal*

Phil Hughes

Issue #14, June 1995

By the time you have reached this page, you may have noticed some changes in the appearance of *Linux Journal*. The changes are more than skin deep and, in this column, I will let you know what is happening.

By the time you have reached this page, you may have noticed some changes in the appearance of *Linux Journal*. The changes are more than skin deep and, in this column, I will let you know what is happening.

The changes have come about both in response to reader feedback and in our attempt to streamline production of the magazine. Up through issue 11, layout was done on a contract basis by an outside person. This interface, both because of the physical location of layout being in a different place than the rest of magazine production and because there was a Macintosh involved, took more time and effort than was practical. With issue 12 we started to move layout in-house and, after a false start or two, we are happy to say all is under control.

Our new layout person is Amy Wood. She has previously worked for a weekly newspaper and the biggest problem we have with her is convincing her that she has a whole month between magazines instead of a week. The layout itself is still done in Quark XPress but it now runs on an MS-Windows system which is connected to our Linux network. The actual interface is handled by Samba (see *LJ* issue 7 for more info on Samba) and is transparent to Amy. For the rest of us, it means that files can be sent to and from that system without the need for *sneaker-net*. (In a future issue we will have an article on how *Linux Journal* is produced using mostly Linux systems.)

Because Amy had so much extra time after finishing up issue 13 she looked at the assorted comments about our layout and along with her own ideas, came up with the current layout. The major changes you should notice are a cleanup

of the page format and the addition of "*continued-from*" lines on the continuation pages of articles which include a keyword so you know what article you are reading. Also, she is putting a lot of effort into minimizing the number of jumps within articles. Please let us know what you think of her work.

The second big change is still under way but should be done by the time you read this. We have established a WWW site, www.ssc.com. As well as having information on SSC products, we will be putting up information from *Linux Journal*. First we will put up the tables of contents and advertisers indices and then add some of the articles from the magazine. Advertisers can request links from their index entry to their web page and, if they don't have their own web site, for a nominal fee we will put their web pages on www.ssc.com. If you are an advertiser and need more information on this, contact Carlie Fairchild at (206) 782-7733.

As well as SSC and *Linux Journal* information, we will have some general Linux information on the site so if you are looking for something to browse try www.ssc.com. This is, of course, a Linux machine. If there are things you think should be added to the site, send e-mail to info@linuxjournal.com.

Finally, with Linux 1.2 out there, more commercial vendors are starting to take Linux seriously. If you see anyone with a commercial interest in Linux, whether it is a vendor interested in making their product work with Linux or a company that is using Linux, please point them our way. The more information of commercial interest we can get in *LJ*, the easier it is for us to convince others that Linux is a viable alternative to other operating systems.

Archive Index Issue Table of Contents

Advanced search

New Products

LJ Staff

Issue #14, June 1995

Metro Link releases Motif 2.0 for Linux, WGS releases Linux Pro 2.2 and more.

Metro Link releases Motif 2.0 for Linux

Metro Link has announced a release of Motif 2.0, OSF's popular graphical user interface. Metro Link Motif 2.0 is ported specifically for use with the new X11R6 version of the X Window System (XFree86 3.I.I. or higher). For developers, Motif 2.0 makes creation of software applications and custom widgets simple. For end users, Motif 2.0 improves their interface performance. The virtual screen support unclutters the workspace by providing alternate locations for chosen windows, while providing greater consistency with PC environments.

Phone 305-938-0283, fax (305) 938-1982, e-mail holly@metrolink.com. Metro Link Incorporated, 4711 North Powerline Road, Ft. Lauderdale, FL 33309.

WGS releases Linux Pro 2.2

WorkGroup Solutions has announced the availability of its new Linux Pro v2.2 on CD-ROM. Users can purchase WGS Linux Pro v2.2 on a single CD-ROM for $19.99, including complete documentation. For more experienced users interested in the latest Linux features, WGS is offering Linux Pro v2.2 in a 3 CD-Rom package for $69. This package includes WGS Linux Pro v2.1 on a single CD, plus two additional CDs that contain images of the tsx-11.mit.edu and sunsite.unc.edu ftp sites. Users who purchase the $69 package have access to comprehensive technical support and receive a WorkGroup Solutions Linux Pro T-shirt.

Contact WorkGroup Solutions on the Internet at World Wide Web address URL: ftp://ftp.wgs.com/pub2/wgs/Filelist, via e-mail at info@wgs.com, fax (303) 699-2793, phone (303) 699-7470.

Mazama Packet Filter

Mazama Software Labs, Inc. has released the Mazama Packet Filter, an easy to install and configure firewall that requires virtually no maintenance. The Mazama Packet Filter filters incoming and outgoing IP packets based on human-recognizable rules. This allows filtering TCP/IP services such as incoming and outgoing ftp, http, gopher, finger, telnet, etc. Specific filter rules based on any TCP/IP packet attributes can be added via the X-based user interface. The Mazama Packet Filter will be generally available on April 15th, at an introductory price of US$750.

Phone (206) 545-1808, e-mail info@mazama.com, on the WWW, please visit the URL http://www.mazama.com/ Mazama Software Labs, 15600 NE 8th St., Suite B1 #544, Bellevue, WA 98008.

Archive Index Issue Table of Contents

Advanced search

Upgrading the Linux Kernel

Mark Komarinski

Issue #14, June 1995

For those who have hesitated to upgrade to the new stable Linux version 1.2, Mark explains how to easily upgrade from version 1.0 (or any subsequent version) to version 1.2 -- and even gives you a little help.

With the release of Linux 1.2.x, there have been a lot of questions on the newsgroups asking what needs to be done to upgrade a stable 1.0.x system to a stable 1.2.x system. Fortunately, this is relatively easy and painless.

One thing you should note is that you should upgrade to the 1.2.1 kernel, especially if you are using firewall software. Version 1.2.0 was somewhat broken in this respect.

The first thing you should realize is that a new program, **bdflush**, is required by the 1.2.x kernel. The program arrived early in the 1.1.x series and replaces **update**, implementing a style of disk caching that is a bit more efficient than the older style. Without this program, your disk caching will not work as well, and you will be warned: "**Warning - bdflush not running**".

The networking tools should also be upgraded. The new versions have been upgraded to support improvements in the kernel networking code. Do note that normal networking programs do not need to be upgraded; only a few special programs, included in a single kit, are affected by the changes.

I have built a package that includes all of these programs. It's called "The Linux Upgrade 1.2", in a file called linuxupgr-1.2.tar.gz. It is available at sunsite.unc.edu and its mirrors in /pub/Linux/system/, and at tsx-11.mit.edu and its mirrors in /pub/linux/binaries/sbin/. Download this package along with the version 1.2.1 kernel, which should be available nearby in the file linux-1.2.1.tar.gz. Sunsite and its mirrors have the kernel in /pub/Linux/kernel/v1.2/, and tsx-11 and its mirrors have the kernel in /pub/linux/sources/system/v1.2/.

Upgrading Your Kernel

1. Back everything up! Back up your system just in case of any problems along the way. In case anything goes wrong, you can always go back to a stable system. At least back up your data files.
2. Make notes about your system. The kind of filesystems you have, extra cards, sound card information, and so on.

   From this point on, you should be the root user with few users logged on.
3. Move the linux kernel tar file to the /usr/src directory. **mv linux-1.2.1.tar.gz /usr/src**
4. Back up the old kernel by moving its directory tree to a directory with another name. For example, if the old kernel is a 1.0.9 kernel, you can (in the /usr/src directory) **mv linux linux-1.0.9**. This way, you still have the source to the old 1.0.9 kernel available.
5. Uncompress and untar the kernel file by running **tar xzvf linux-1.2.1.tar.gz** You should see a list of files being sent to the screen. They should all begin with "linux/".
6. Change to the linux directory and run **make config**. You will be offered options, for which you will need the information you collected from step 2. In many cases, you can take the defaults, unless you have a specialized system. Issue 7 of *Linux Journal* had an article, "Linux Performance Tuning for the Faint of Heart", which specifically covers how to compile new kernels.

   One specific item that is not covered in that article is the new IDE driver in Linux 1.2. In the Linux kernel source, there is a file drivers/block/README.ide that you should read if you have a large IDE drive, multiple IDE controllers, or ATAPI IDE CD-ROM drives. Nearly everyone will want to answer "Use new IDE driver for primary/secondary [interface]" with yes. The only reason not to is if you use MFM or RLL drives, since the new IDE driver supports only IDE drives.
7. Once the configuration is ready, make the dependencies needed by gcc to build the kernel correctly. Run **make dep**.
8. At this point, if you're using lilo, you can run **make zlilo** and the makefile will automatically begin building the kernel and installing the new kernel in LILO. If you boot off a floppy drive, you can just run **make zdisk**, and put a blank formatted 1.4 (or 1.2) MB disk in the drive and makefile will build the kernel and install it on your floppy.

   NOTE: My preference is to install first to a diskette, and boot from that. If there are no problems, then I go back and run **make zlilo**. Another option is copying the current Linux kernel to another filename (such as vmlinuz1.0.9) and making a new LILO entry for that file. If you reboot with the first kernel and have problems, you can reboot again, get into LILO,

and boot the older kernel. If you feel comfortable using LILO, this may be easier for you.

9. You will also need to tell the kernel file to mount the root device read-only so that it can run fsck if necessary. If you're using a floppy, run **rdev -R /dev/fd0 1**. If you're using LILO, run **rdev -R /vmlinuz 1**. Using the capital R here is important. If you are using LILO, another option is to use the **read-only** option for the boot image. This is covered in the excellent LILO documentation.
10. Once the kernel is built, install the Linux Upgrade Package. From /usr/src, run **tar xzvf linuxupgr-1.2.tar.gz**. This will create a directory called upgrade which has the bdflush and other assorted files in it. Read the README file for instructions on installing the upgrade files.
11. Reboot the machine. If you did everything correctly, you should have a successfully running 1.2.1 kernel. If you do run into problems and you are using a floppy disk, you can pop the floppy out of the drive and reboot with the old kernel.

Misc. Other Upgrades

There are a few other packages you may want to upgrade at this point. Among them are **gcc**, libraries, and the X Window System. However, the old versions will work with the newer kernels, and upgrading is not necessary for running Linux 1.2.1.

If you have any comments about this article or suggestions for future articles, please e-mail me at komarimf@craft.camp.clarkson.edu

Mark Komarinski graduated from Clarkson University (in very cold Potsdam, NY) with a degree in computer science and technical communication. He now lives in Troy, NY, spending much of his free time working for the Department of Veterans Affairs where he is a programmer.

Archive Index Issue Table of Contents

Advanced search

The Linux keyboard driver

Andries E. Brouwer

Issue #14, June 1995

Our Kernel Korner series continues with an article describing the linux Keyboard driver. This artilce is not for “Kernel Hackers” only—in fact, it will be most useful to those who wish to use their own keyboard to its fullest potential, and those who want to write programs to take advantage of the many features in the Linux keyboard driver.

When you press a key on the console keyboard, the corresponding character is not simply added to the **tty** (generic terminal handling) input buffers as if it had come in over a serial port. A lot of processing is required before the kernel knows what the correct characters are. Only after processing can the generic tty code, which handles all interactive terminal devices, take over.

Roughly speaking, the picture is this: the keyboard produces scancodes, the scancodes are assembled into keycodes (one unique code for each key), and keycodes are converted to tty input characters using the kernel keymaps. After that, the normal `stty' processing takes place, just as for any other terminal.

Scancodes First

The usual PC keyboards are capable of producing three sets of scancodes. Writing 0xf0 followed by 1, 2 or 3 to port 0x60 will put the keyboard in scancode mode 1, 2 or 3. Writing 0xf0 followed by 0 queries the mode, resulting in a scancode byte 0x43, 0x41 or 0x3f from the keyboard. (Don't try this at home, kids. If you are not very careful, you will end up in a situation where rebooting is the only way out—and control-alt-delete will not be available to shut the computer down correctly. See the accompanying listing of kbd_cmd.c for details.)

Scancode mode 2 is the default. In this mode, a key press usually produces a value s in the range 0x01-0x5f and the corresponding key release produces s+0x80. In scancode mode 3, the only key releases that produce a scan code

are of both Shift keys, and the left Ctrl and Alt keys; for all other keys only the key presses are noted. The produced scancodes are mostly equal to those for scancode mode 2.

In scancode mode 1 most key releases produce the same values as in scancode mode 2, but for key presses there are entirely different, unrelated values. The details are somewhat messy.

A program can request the raw scancodes by

```
ioctl(0, KDSKBMODE, K_RAW);
```

For example, **X**, **dosemu**, **svgadoom**, and **showkey -s** do this. The default keycode translation mode is restored by

```
ioctl(0, KDSKBMODE, K_XLATE);
```

See the keyboard FAQ (in kbd-0.90.tar.gz) for some advice about how to get out of raw scancode mode from the shell prompt. (At a shell prompt the commands **kbd_mode [-s|-k|-a|-u]** will set the keyboard mode to scancode mode, keycode mode, translation (`ASCII') mode and Unicode mode, respectively. But it is difficult to type this command when the keyboard is in raw scancode mode.)

Scancodes to Keycodes

Life would have been easy had there been a 1-1 correspondence between keys and scancodes. (And in fact there is, in scancode mode 3, but that does not suffice for Linux, since X requires both the key press and the key release events.)

But as it is, a single key press can produce a sequence of up to six scancodes, and the kernel has to parse the stream of scancodes and convert it into a series of key press and key release events. To this end, each key is provided with a unique keycode k in the range 1-127, and pressing key k produces keycode k, while releasing it produces keycode k+128. The assignment of key codes is in principle arbitrary (and has no relation to the key codes used by X), but at present the key code equals the scan code for those keys that produce a single scancode in the range 0x01-0x58.

The parsing works by

- recognizing the special sequence 0xe1 0x1d 0x45 0xe1 0x9d 0xc5 produced by the Pause key

- throwing out any fake Shift-down and Shift-up codes, inserted by the keyboard to make the kernel believe that you pressed Shift to undo the effect of NumLock
- recognizing scancode pairs 0xe0 s
- recognizing single scancodes s.

Since s can take 127 values (0 is a keyboard error condition, and the high order bit indicates press/release) this means that parsing could result in 1+127+126=254 distinct keycodes. However, at present keycodes are restricted to the range 1-127 and we have to work a little to make things fit. (No doubt sooner or later keycodes will be integers instead of 7-bit quantities, and the keymaps will be sparse, but for the time being we can avoid that—since to my knowledge no actual PC keyboard has more than 127 keys.) So, there are small tables that assign a keycode to a scancode pair 0xe0 s or to a single scancode in the range 0x59-0x7f. In the default setting everything works for most current keyboards, but in case you have some strange keyboard, you can get the kernel to recognize an otherwise unrecognized key by filling an entry in these tables using the KDSETKEYCODE ioctl; see setkeycodes(8).

Two keys are unusual in the sense that their keycode is not constant, but depends on modifiers. The PrintScrn key will yield keycode 84 when combined with either Alt key, but keycode 99 otherwise. The Pause key will yield keycode 101 when combined with either Ctrl key, but keycode 119 otherwise. (This has historic reasons, but might change, to free keycodes 99 and 119 for other purposes.)

At present there is no way to tell X about strange key(board)s. The easiest solution would be to make X use keycodes instead of scancodes, so that the information about strange keys and the scancodes they produce is located a single place.

A program can request to get keycodes by doing

```
ioctl(0, KDSKBMODE,
K_MEDIUMRAW);
```

For example, showkey does this. Warning: the details of the function of both the **KDSETKEYCODE** ioctl and the **K_MEDIUMRAW** keyboard mode are likely to change in the future.

Keymaps

Keycodes are converted to keysymbols by looking them up on the appropriate keymap. There are eight possible modifiers (shift keys), and the combination of currently active modifiers and locks determines the keymap used.

Thus, what happens is approximately:

```
int shift_final = shift_state ^ kbd->lockstate;
 ushort *key_map = key_maps[shift_final];
 keysym = key_map[keycode];
```

The eight modifiers are known as Shift, AltGr, Control, Alt, ShiftL, ShiftR, CtrlL and CtrlR. These labels have no intrinsic meaning, and the modifiers can be used for arbitrary purposes, except that the keymap for the Shift modifier determines the action of CapsLock (and that the Shift key partially suppresses keyboard application mode). By default Shift is bound to both Shift keys and Control keys and Alt and AltGr are bound to the left and right Alt keys. The remaining four modifiers are unbound in the default kernel. X is able to distinguish ShiftL and ShiftR, etc.

Thus, there are 256 possible keymaps—for plain symbols, for Shift+symbol, for Ctrl+AltL+Shift+symbol, etc. Usually, not all of the keymaps will be allocated (combinations with more than three modifiers are rather unusual), and in fact the default kernel allocates only 7 keymaps, namely the plain, Shift, AltR, Ctrl, Ctrl+Shift, AltL and Ctrl+AltL maps. You can allocate more keymaps simply by filling some of their entries using **loadkeys(1)**.

Key # symbols

Key symbols are **shorts**, i.e., they consist of two bytes. In Unicode mode, this short is just the 16-bit value returned—or, to be precise, the returned byte string is the UTF-8 representation of this Unicode character. The keyboard is put into Unicode mode by

```
ioctl(0, KDSKBMODE, K_UNICODE);
```

When not in Unicode mode, the high order byte is viewed as a type, and the low order byte as a value, and we do:

```
type = KTYP(keysym);
 (*key_handler[type])(keysym & 0xff, up_flag);
```

The type selects a function from the array key_handler:

```
static k_hand key_handler[16] = {
     do_self, do_fn, do_spec, do_pad, do_dead,
     do_cons, do_cur, do_shift, do_meta, do_ascii,
     do_lock, do_lowercase, do_ignore, do_ignore,
```

```
    do_ignore, do_ignore
};
```

1. **do_self**, commonly used for ordinary keys, just returns the given value, after possibly handling pending dead diacriticals.
2. **do_fn**, commonly used for function keys, returns the string **func_table[value]**. Strings can be assigned using loadkeys(1).
3. **do_spec** is used for special actions, not necessarily related to character input. It does **spec_fn_table[value]();**, where

   ```
   static void_fnp spec_fn_table[] = {
   do_null, enter, show_ptregs, show_mem,
   show_state, send_intr, lastcons, caps_toggle,
   num, hold, scroll_forw, scroll_back, boot_it,
   caps_on, compose, SAK, decr_console,
   incr_console, spawn_console, bare_num
    };
   ```

   The associated actions (and their default key binding) are:
   - Return (Enter): return a **CR** and if **VC_CRLF** mode set also a **LF**. One sets/clears **CRLF** mode by sending **ESC [20 h** or **ESC [20 l** to the console.
   - Show_Registers (AltR-ScrollLock): print the contents of the CPU registers.
   - Show_Memory (Shift-ScrollLock): print current memory usage.
   - Show_State (Ctrl-ScrollLock): print the process tree.
   - Break (Ctrl-Break): send a Break to the current tty.
   - Last_Console (Alt-PrintScrn): switch to the last used console.
   - Caps_Lock (CapsLock): toggle the CapsLock setting.
   - Num_Lock (NumLock): in keyboard application mode: return **ESC O P**; otherwise, toggle the NumLock setting. One sets/clears keyboard application mode by sending **ESC = or ESC >** to the console. (See also Bare_Num_Lock below.)
   - Scroll_Lock (ScrollLock): stop/start tty—roughly equivalent to **^S/^Q**.
   - Scroll_Forward (Shift-PageDown): scroll console down.
   - Scroll_Backward (Shift-PageUp): scroll console up. These two functions are implemented by using the memory on the video card, and provide only a very limited scrollback facility. Moreover, all scrollback information is lost when you switch virtual consoles. So, for real scrollback use a program-like screen.
   - Boot (Ctrl-AltL-Del): reboot. If you press Ctrl-AltL-Del (or whatever key was assigned the keysym Boot by loadkeys) then either the machine reboots immediately (without sync), or **init** is sent a **SIGINT**. The former behavior is the default. The default can be changed by root, using the system call **reboot():** see **ctrlaltdel(8)** and **init(8)**.

Some versions of init change the default. What happens when init gets SIGINT depends on the version of init used—often it will be determined by the pf (stands for powerfail) entry in /etc/inittab, which means that you can run an arbitrary program. In the current kernel, Ctrl-AltR-Del is no longer by default assigned to Boot, only Ctrl-AltL-Del is.

Sometimes when init hangs in a disk wait (and syncing is impossible) it can be useful to say ctrlaltdel hard, which may allow you to force a reboot without power cycling or pressing the reset button.

- aps_On (none): set CapsLock.
- ompose (Ctrl-.): start a compose sequence. The two following characters will be combined. This is a good way to get accented characters that you only rarely need. For example, **Ctrl-.><,><c** will produce a c-cedilla, and **Ctrl-.&gt:<a><e** the Danish letter æ. Precisely which combinations combine to what character; will show **dumpkeys(1)**, loadkeys(1) will set combinations.
- SAK (none): Secure Attention Key. This is supposed to kill all processes related to the current tty, and reset the tty to a known default state. It is not completely implemented—it is not quite clear what resetting the keyboard/console should do to the fonts and keymaps. The easiest solution is to send a signal to some trusted daemon, and let it reset keyboard and console as desired. In this way we obtain something closely related to the Spawn_Console function below.
- Decr_Console (AltL-LeftArrow): switch to the virtual console that precedes the current console in the cyclic order.
- Incr_Console (AltL-RightArrow): switch to the virtual console that follows the current console in the cyclic order.
- Spawn_Console or KeyboardSignal (AltL-UpArrow): send a specified process a specified signal. I use this to signal init that it should create a fresh virtual console for me.
- Bare_Num_Lock (Shift-NumLock): toggle the NumLock setting (regardless of keyboard application mode).

As long as no new releases of init and loadkeys have come out, you can play with this by using loadkeys and starting the program spawn_console:

```
% loadkeys >> EOF
 alt keycode 103 = 0x0212
 EOF
 % spawn_console &
```

Of course, if you put this in /etc/rc.local, you would probably want to start **getty** instead of **bash**.

4. **do_pad**, commonly used for the keypad keys. In keyboard application mode this produces some three-character string **ESC O X** (with variable X depending on the key), provided that Shift is not pressed simultaneously. Otherwise, when NumLock is on, we get the symbol printed on the key (0123456789.+ -/* and CR).

   Finally, if NumLock is not on, the four arrow keys yield **ESC [X** (with **X=A**, **B**, **C**, or **D**p when not in cursor key mode, and ESC O X otherwise, while the remaining keys are treated as function keys, and yield the associated string. For the middle key (keypad-5) we find four possibilities:
   - in keyboard application mode (unshifted), ESC O u
   - in keyboard application mode, shifted, without NumLock, ESC O G
   - otherwise, without NumLock, ESC [G
   - but with NumLock, 5.

   If you think this is unnecessarily complicated, I agree. It is a messy combination of VT100 and DOS keyboard behavior. However, so far suggestions for change have met with too much resistance.
5. **do_dead** is used for "dead keys" that provide the following key with a diacritical. By default there are no dead keys. One may define keys producing a dead grave, acute, circumflex, tilde, or diaeresis. How a dead key combines with a following key is specified using the compose mechanism discussed above.
6. **do_cons** is used for switching consoles. By default the combinations (Ctrl-)AltL-F*n* switch to virtual console *n* for *n* in the range 1-12, and AltR-Fn switches to console *n*+12 for these same *n*.
7. **do_cur** handles cursor keys. One gets either **ESC [X** or **ESC O X** (with *X* one of **A, B, C**, or **D**) depending on the cursor key mode. (One sets or clears cursor key mode by sending **ESC [? 1 h** or **ESC [? 1 l** to the console.)
8. **do_shift** maintains the shift state (the up/down state of the modifier keys).
9. **do_meta** is commonly used for ordinary keys combined with AltL. If the keyboard is in metamode, this will yield a pair **ESC x**; otherwise **x | 0x80** is produced, where *x* is the key pressed in both cases. (You can set or clear metamode using the tiny utility **setmetamode(1)**.)
10. **do_ascii** is used to construct given codes: press AltL, type a decimal code on the keypad, and release AltL. This yields the character with the given code. In Unicode mode the same works in hexadecimal: press AltR, type a hexadecimal code on the keypad, possibly using the ordinary a, b, c, d, e, and f keys, and release AltR. This yields the Unicode symbol with the code given.
11. **do_lock** toggles the state of the corresponding modifier key lock. (Recall the line we saw above: **shift_final = shift_state ^ kbd-<lockstate**.) Thus, if you have your Cyrillic keys under combinations with AltR, you can use AltR

together with other keys to get only a few Cyrillic symbols, but should type AltGr_Lock if you plan to type a longish Cyrillic text. (Note that the right Alt key, that I called AltR here, is usually known as AltGr.)

12. **do_lowercase** is used for the handling of CapsLock. Note that CapsLock is different from ShiftLock. With ShiftLock, a digit 4 will be turned into a dollar sign (for default keyboard layout), but CapsLock will only affect lower case letters, and turn them into the corresponding upper case letters. Type 11 is equivalent to type 0, with the added information that the symbol may be affected by CapsLock (and the resulting character is the one that would have resulted from pressing Shift).

As already mentioned, almost all of this can be changed dynamically by use of loadkeys(1). The current state is dumped by dumpkeys(1). A list of known symbols is provided by dumpkeys -l. The keycodes associated with the various keys can be found using showkey(1). These and many other utilities for keyboard and console can be found in kbd-0.90.tar.gz on ftp.funet.fi and its mirror sites.

Using loadkeys

Use loadkeys to change the code produced by the BackSpace key from Delete to BackSpace:

```
% loadkeys
keycode 14 = BackSpace
```

Assign the string “emacs\n” to the function key F12, and “rm *~\n” to Shift-F12 (the keycode 88 was found using showkey; the F66 is a random unused function key symbol):

```
% loadkeys
 keycode 88 = F12
 shift keycode 88 = F66
 string F12 = "emacs\n"
 string F66 = "rm *~\n"
```

Create the compose combination that will compose | and S into $:

```
% loadkeys
 compose '|' 'S' to '$'
 compose 'S' '|' to '$'
```

Reset to some default state:

```
% loadkeys -d
```

Sequel

After the above handling, the obtained character(s) are put in the raw tty queue. Depending on the mode of the tty, they will be processed and transferred to the cooked tty queue. (Don't confuse raw mode as **stty** knows it, with the raw scancode mode discussed above.) Finally, the application will get them when it does **getchar();**.

Andries Brouwer, aeb@cwi.nl, has used Unix for various mathematical, linguistic, and playful purposes the past 20 years or so. He might be known to some for the first net release of **hack**.

Archive Index Issue Table of Contents

Advanced search