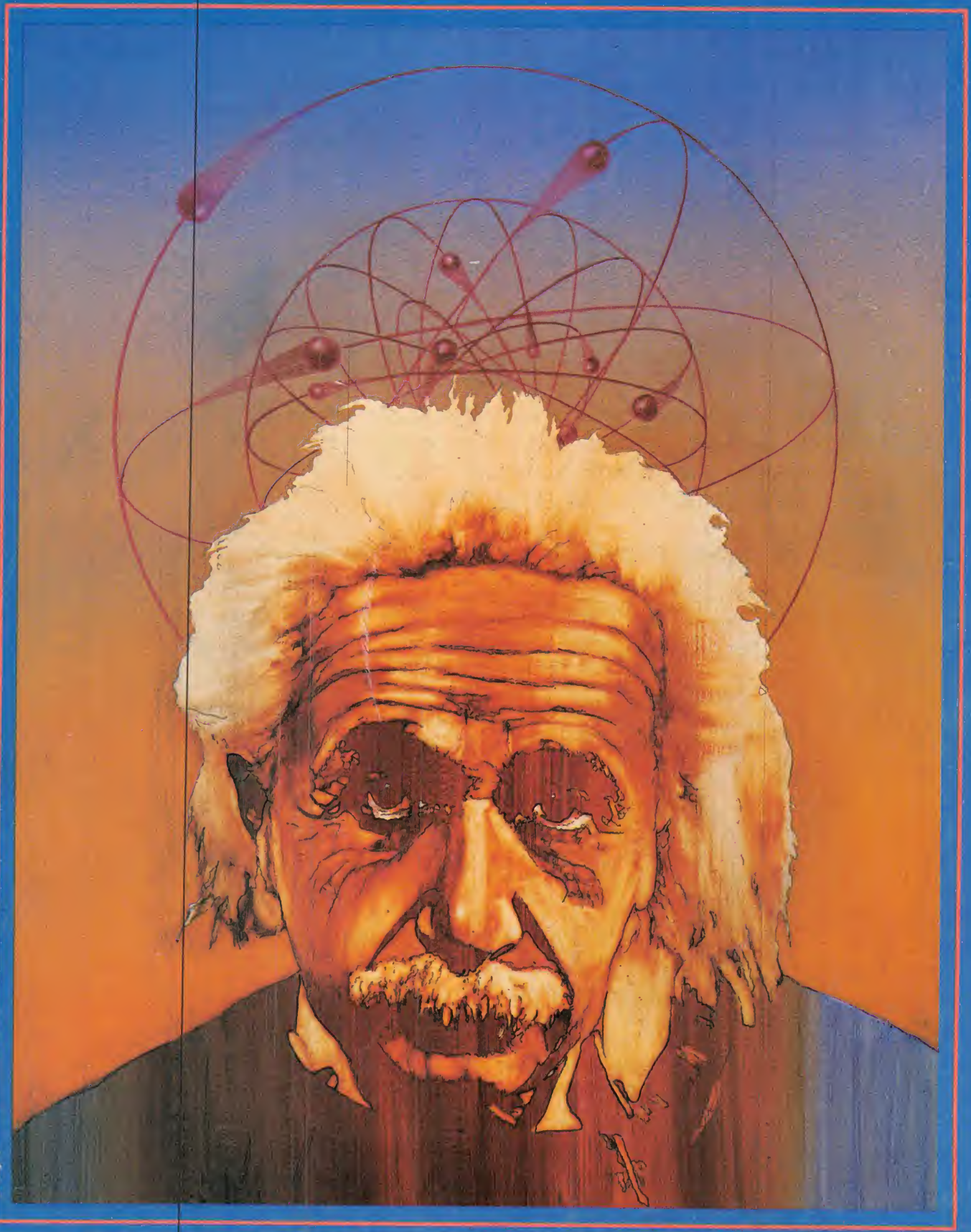
UNIX™ REVIEW

THE PUBLICATION FOR THE UNIX™ COMMUNITY

NOVEMBER 1985 $3.95



SCIENTIFIC APPLICATIONS

HANDLE™

Modular.
Integrated.
Now.

Handle Writer/Spell™
Word processing with integrated spelling correction and verification.

Handle Calc™
Spreadsheet with up to 32,000 rows and columns. Conditional and iterative recalculation.

The **Handle Office-Automation Series** is a powerful set of modular, integrated software tools developed for today's multiuser office environment. **Handle** application modules can be used stand-alone or combined into a fully integrated system.

The **Handle Office-Automation Series** modules offer:

- Ease of Use and Learning
- Insulation from **UNIX**
- Data Sharing Between Multiple Users
- Data Integration Between Modules
- Data Sharing with Other Software Products
- Sophisticated Document Security System

Handle Technologies, Inc.

Corporate Office
6300 Richmond
3rd Floor
Houston, TX 77057
(713) 266-1415

Sales and Product Information
850 North Lake Tahoe Blvd.
P.O. Box 1913
Tahoe City, CA 95730
(916) 583-7283

TM—HANDLE, HANDLE HOST, HANDLE WRITER, HANDLE SPELL, HANDLE WRITER/SPELL, and HANDLE CALC ARE TRADEMARKS OF HANDLE TECHNOLOGIES, INC.
TM—UNIX IS A TRADEMARK OF AT&T BELL LABORATORIES.

Circle No. 255 on Inquiry Card

How to go from UNIX to DOS without compromising your standards.

It's easy. Just get an industry standard file access method that works on both.

C-ISAM™ from RDS.

It's been the UNIX™ standard for years (used in more UNIX languages and programs than any other access method), and it's fast becoming the standard for DOS.

Why?

Because of the way it works. Its B+ Tree indexing structure offers unlimited indexes. There's also automatic or manual record locking and optional transaction audit trails. Plus index compression to save disk space and cut access times.

How can we be so sure C-ISAM works so well? We use it ourselves. It's a part of INFORMIX,® INFORMIX-SQL and File-it!,™ our best selling database management programs.

For an information packet, call (415) 322-4100. Or write RDS, 4100 Bohannon Drive, Menlo Park, CA 94025.

You'll see why anything less than C-ISAM is just a compromise.

RDS™

RELATIONAL DATABASE SYSTEMS, INC.

© 1985, Relational Database Systems, Inc. UNIX is a trademark of AT&T
INFORMIX is a registered trademark and RDS, C-ISAM and File-It! are trademarks of Relational Database Systems, Inc.

Circle No. 269 on Inquiry Card

How we improved Structured Query Language.

Actually, we didn't change a thing.

We just combined it with the best relational database management system.

Introducing INFORMIX®-SQL.

It runs on either MS™-DOS or UNIX™ operating systems. And now with IBM's SQL as part of the program, you can ask more of your database. Using the emerging industry-standard query language.

To make your job easier, INFORMIX-SQL comes with the most complete set of application building tools. Including a full report writer and screen generator. Plus a family of companion products that all work together.

Like our embedded SQLs for C and COBOL. So you can easily link your programs with ours. File-it!™ our easy-to-use file manager. And C-ISAM™, the de facto

INFORMIX is a registered trademark and RDS, C-ISAM and File-it! are trademarks of Relational Database Systems, Inc. IBM, UNIX and MS are trademarks of International Business Machines Corporation, AT&T and Microsoft, respectively. © 1985, Relational Database Systems, Inc.

standard ISAM for the UNIX operating system. It's built into all our products, but you can buy it separately.

And when you choose RDS, you'll be in the company of some other good companies. Computer manufacturers including AT&T, Northern Telecom, Altos and over 60 others. And major corporations like Anheuser Busch and The First National Bank of Chicago.

Which makes sense. After all, only RDS offers a family of products that work so well together. As well as with so many industry standards.

So call us for a demo, a manual and a copy of our Independent Software Vendor Catalog. Software vendors be sure to ask about our new "Hooks" software integration program. Our number: 415/322-4100.

Or write RDS, 4100 Bohannon Drive, Menlo Park, CA 94025.

And we'll show you how we took a good idea and made it better.

RDS™

RELATIONAL DATABASE SYSTEMS, INC.

UNIX™ REVIEW

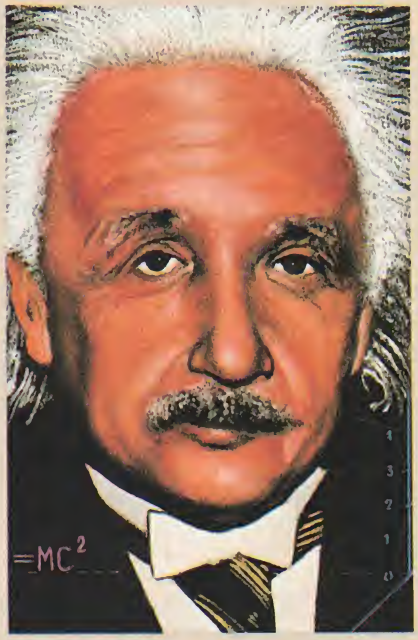
THE PUBLICATION FOR THE UNIX COMMUNITY

Volume 3,
Number 11
November 1985

DEPARTMENTS:

Cover art by Heda Majlessi

FEATURES:



SCIENTIFIC APPLICATIONS

UNIX REVIEW (ISSN-0742-3136) is published monthly by REVIEW Publications Co. It is a publication dedicated exclusively to the needs of the UNIX community. Second class postage paid at Renton, WA 98055 and at additional mailing offices. POSTMASTER: Please send Form 3579 to UNIX REVIEW, 500 Howard Street, San Francisco, CA 94105.

Entire contents copyright 1985. All rights reserved and nothing may be reproduced in whole or in part without prior written permission from UNIX REVIEW.

Subscriptions to UNIX REVIEW are available at the following annual rates (12 issues): US$28 in the US; US$35 in Canada; US$48 in all other countries/surface mail; US$85 in all other countries/air mail. Correspondence regarding editorial (press releases, product announcements) and circulation (subscriptions, fulfillment, change of address) should be sent to 500 Howard Street, San Francisco, CA 94105. Telephone 415/397-1881. Correspondence regarding dealer sales should be sent to 901 South 3rd Street, Renton, WA 98055. Telephone 206/271-9605.

Letters to UNIX REVIEW or its editors become the property of the magazine and are assumed intended for publication and may so be used. They should include the writer's full name, address and home telephone number. Letters may be edited for the purpose of clarity or space. Opinions expressed by the authors are not necessarily those of UNIX REVIEW.
UNIX is a trademark of AT&T Bell Laboratories, Inc. UNIX REVIEW is not affiliated with AT&T Bell Laboratories.

PUBLISHER:
Pamela J. McKee
ASSOCIATE PUBLISHERS:
Ken Roberts, Scott Robin
EDITORIAL DIRECTOR:
Stephen J. Schneiderman
EDITOR:
Mark Compton
ASSOCIATE EDITOR:
David Chandler
EDITORIAL ADVISOR:
Dr. Stephen R. Bourne, Consulting Software Engineer, Digital Equipment Corporation
EDITORIAL REVIEW BOARD:
Dr. Greg Chesson, Chief Scientist, Silicon Graphics, Inc.
Larry Crume, President/Managing Director, AT&T UNIX Pacific Co., Ltd.
Ted Dolotta, Senior Vice President of Technology, Interactive Systems Corporation
Ian Johnstone, Project Manager, Operating Software, Sequent Computer Systems
Bob Marsh, Chairman, Plexus Computers
John Mashey, Manager, Operating Systems, MIPS Computer Systems
Robert Mitze, Department Head, UNIX Computing System Development, AT&T Bell Labs
Deborah Scherrer, Computer Scientist, Mt. Xinu
Jeff Schriebman, President, UniSoft Systems
Rob Warnock, Consultant
Otis Wilson, Manager, Software Sales and Marketing, AT&T Information Systems
HARDWARE REVIEW BOARD:
Gene Dronek, Director of Software, Aim Technology
Doug Merritt, Consultant
Richard Morin, Consultant, Canta Forda Computer Laboratory
Mark G. Sobell, Consultant
SOFTWARE REVIEW BOARD:
Eric Allman, Principal Systems Engineer, Britton Lee, Inc.
Ken Arnold, Consultant, UC Berkeley
Jordan Mattson, Programmer, UC Santa Cruz
Dr. Kirk McKusick, Research Computer Scientist, UC Berkeley
Doug Merritt, Consultant
Mark G. Sobell, Consultant
CONTRIBUTING EDITOR:
Ned Peirce, Systems Analyst, AT&T Information Systems
PRODUCTION DIRECTOR:
Nancy Jorgensen
PRODUCTION STAFF:
Cynthia Grant, Tamara V. Heimarck, Florence O'Brien, Denise Wertzler
BUSINESS MANAGER:
Ron King
CIRCULATION DIRECTOR:
Wini D. Ragus
CIRCULATION MANAGER:
Jerry M. Okabe
MARKETING MANAGER:
Donald A. Pazour
OFFICE MANAGER:
Tracey J. McKee
TRAFFIC:
Tom Burrill, Dan McKee, Corey Nelson
NATIONAL SALES OFFICES:
500 Howard St.
San Francisco, CA 94105
(415) 397-1881
Regional Sales Manager:
Colleen M. Y. Rodgers
Sales/Marketing Assistant:
Anmarie Achacoso

370 Lexington Ave.
New York, NY 10017
(212) 683-9294
Regional Sales Manager:
Katie A. McGoldrick

BPA membership applied for in March, 1985.

VIEWPOINT

The life cycle

It has been observed that since UNIX now has a fair amount of market momentum, it must be well past its prime technically. Common wisdom, after all, holds that public acceptance and heavy press coverage are the surest signs of obsolescence.

Given this perspective, however, it's difficult to assess the role UNIX might have in the scientific community. Scientists certainly would not be quick to say that the system's best years are behind it. They know that if UNIX is to make a significant contribution in their field, it will need to achieve a much greater penetration than it currently enjoys.

This, of course, has given rise to the question: is UNIX, in fact, suitable? The logic in this is good, but the question is bad. For the last 15 years, UNIX has won hearts and minds in almost every other realm by virtue of its portability and flexibility. Only the most facile mind can imagine the array of esoteric UNIX adaptations already in use. The system has survived as long as it has largely because of its ready acceptance of change.

So we return to the question: can UNIX be adapted for scientific use? Yes, of course—but probably not without a price. The questions that demand answers are: what cost-effective adaptations might be made and how might UNIX offer solutions that are better than those already available to scientists?

This last question is especially intriguing since it takes inertia into account. Scientists, like people in other professions, have a vested interest in the status quo. Apart from explorers with masochistic tendencies, most people shun the pain of transition unless they can be assured that the grass is definitely greener on the other side—substantially greener, in fact.

UNIX has yet to demonstrate to scientists that its solutions are that much better than the ones offered by VMS. Indeed, some in the scientific community question whether UNIX is better at all. At the root of this doubt lies the Fortran question—a matter that Lawrence Berkeley Lab's Joe Sventek wrestles with in the lead article of this issue.

Bob Goff follows with an account detailing some of the data analysis strengths brought to bear by UNIX. As a researcher who has spent much of his life manipulating seismic data, Goff speaks from experience.

The tools offered by UNIX are yet another lure deserving attention. One tool in particular, the S system, was specifically designed for data analysis. Rick Becker and John Chambers, the gentlemen responsible for the system's development, describe it and discuss how UNIX influenced this design.

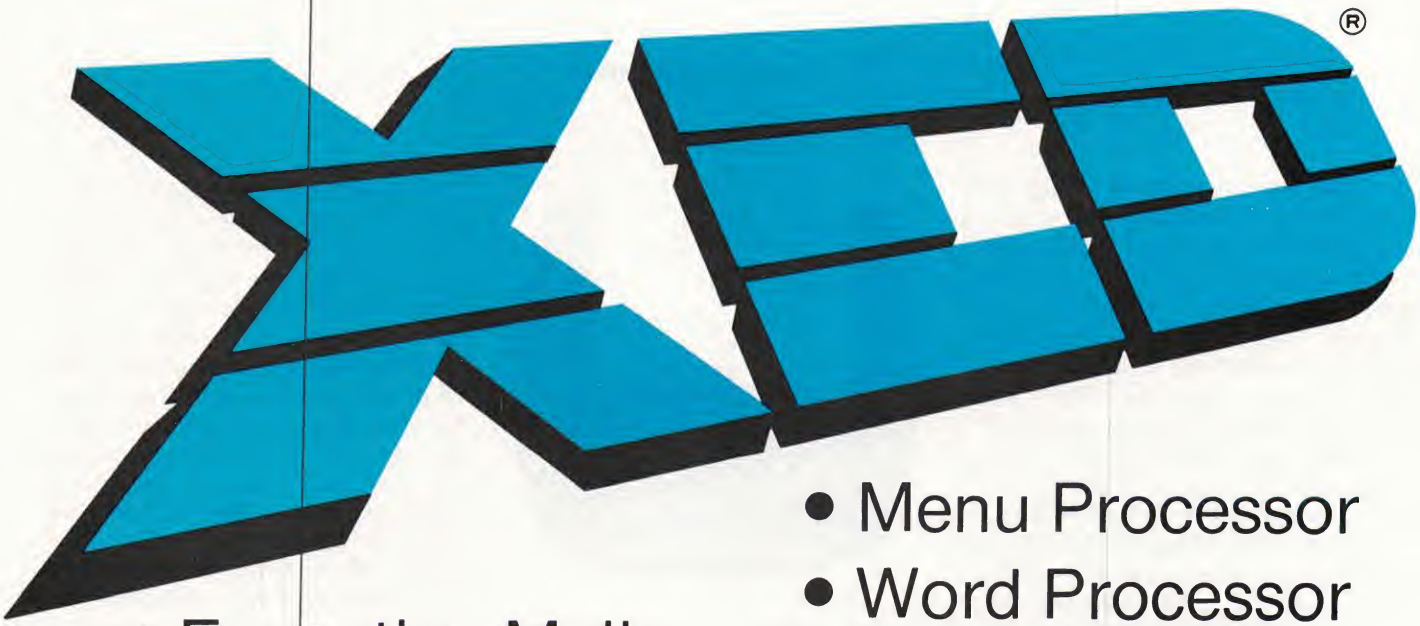
The issue then forges into a bugaboo topic—real time. Some critics say UNIX can't handle real-time applications effectively. Clem Cole of MASSCOMP disagrees, and he offers seven good reasons why.

The theme closes with an interview of Steve Wallach, the man who helped generate the "Crayette" wave with his design of the Convex C-1. If Wallach's name sounds especially familiar, it's probably because you've read about him in *The Soul of a New Machine*. The questions he addresses come from Rob Warnock, himself a systems architect.

If all this seems to suggest that adventures still lie ahead for UNIX, so be it. In the scientific realm at least, UNIX still has many frontiers left to cross.

Mark Compton

The First Name In Integrated Office Automation Software



- Menu Processor
- Word Processor
- Forms/Data Base
- Spreadsheet
- Executive Mail
- Telephone Directory

Certified and Deliverable Since 1981

XED was the first independent software company to introduce a Unix WP package and achieved early success by selling to the government and international market (XED is the only Unix WP package to meet government specifications). Worldwide sales of XED rank Computer Methods first in both sales and units installed in 1984.

XED® INTEGRATED OFFICE SOFTWARE

Box 3938 • Chatsworth, CA 91313 U.S.A. • (818) 884-2000
FAX (818) 884-3870 • Intl. TLX 292 662 XED UR

XED is a registered trademark of CCL Datentechnik AG
UNIX is a trademark of AT & T Bell Laboratories, Inc.

Circle No. 264 on Inquiry Card

THE MONTHLY REPORT

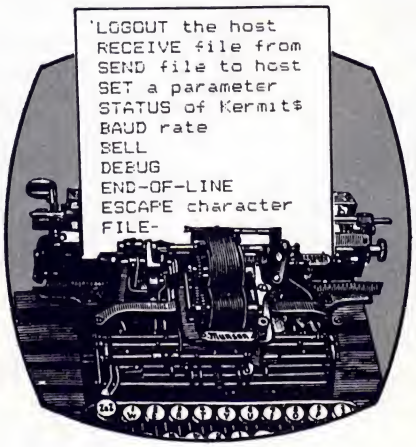
No simple answers

by David Chandler

Two friends were chatting one day. One, a casual fellow with a penchant for keeping things simple, was commenting on the other's verbosity. "Richard", he said, "you're always launching into some diatribe when all I want is a simple response. Can't you ever give me a straight answer?" To this the friend replied, "Well, yes and no. Let me explain. . . ."

As mentioned in last month's *Report*, AT&T and Sun Microsystems, Inc., announced in September a major technology-sharing agreement whereby technical representatives from both companies will work together "to facilitate convergence" of System V and the 4.2BSD-based Sun OS. At first glance, the agreement seems to hold great potential for contributing to the evolution of UNIX as a computer industry standard. Further study, however, reveals that there are certain portions of the announcement which are quite significant, and certain others which are less so. While working to avoid verbosity, an explanation is in order.

There was commotion in the UNIX community when the announcement was first made—and for good reason. Perhaps the greatest excitement was felt by those who wish for UNIX to become the *official* standard that many say it already is unofficially. Industry watchers thus were



stirred when UNIX giants AT&T and Sun signed an agreement. Add to this the opening line of the fact sheet Sun distributed along with its press release: "Sun and AT&T have agreed to work together to converge the two major UNIX standards into a single version." Further fanning of the flames came from the industry press, as evidenced by the front page story in *Computerworld* that cried out, "AT&T, Sun to Redo UNIX". Such stories may not be as racy as amendments to the Ten Commandments, but they do raise eyebrows.

The facts as presented in the announcement of the agreement are these: Sun and AT&T will incorporate a "reasonable superset" of both System V and Sun OS into a single, AT&T-endorsed, enhanced version of System V. The resulting package will be available from both companies—Sun will offer an implementation of the common interface on Sun workstations (by summer 1986), and AT&T will license it in a future enhanced version of System V. (Estimates from Sun hold that the process at AT&T may take as long as two years.) The new system will continue to run the existing base of System V applications and will provide the networking services that previously have been available only in 4.2BSD systems. (All of this, of course, supports Bruce Borden's thesis that, "The way a standard develops is from the implementation backward as opposed to the definition forward." Borden, the manager of engineering at Silicon Graphics, Inc., should know—he's been in the UNIX game since the Edition 4 days.)

Presenting this information, however, raises more questions than it answers. What will the Sun-AT&T convergence include? What will it exclude? Which company will contribute what? AT&T and Sun are known for having different views on networking—what does this agreement say about that?

The first two questions—what will and won't be included in the system—are loaded ones, and company sources decline to be specific in responding. This indicates either that they are (under-

COSI

T A N G O™

Use Tango to:

- Connect IBM and compatible PC's running DOS to UNIX systems.
- Offload processing to PC's.
- Control data and applications on remote PC's.
- Distribute processing between UNIX and PC's.

Buy Tango for:

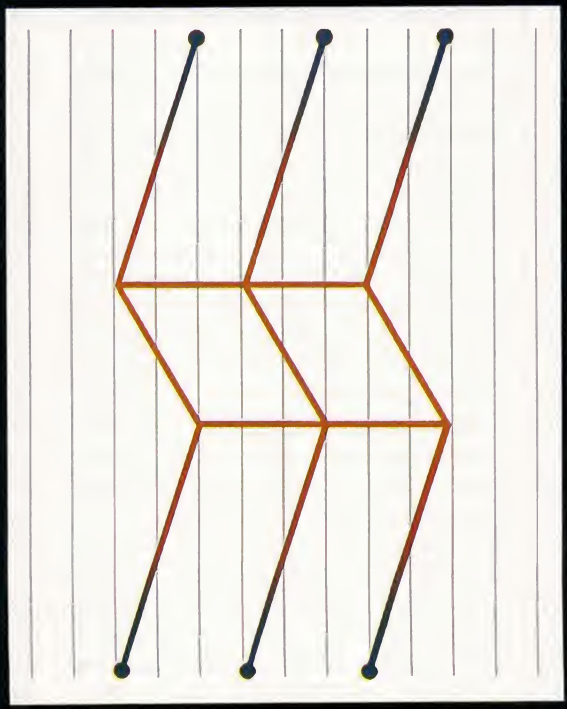
- Execution of DOS programs on the PC under UNIX control.
- Simple elegant file transfer under error correcting protocol.
- DEC, IBM, and Tektronix (graphics) terminal emulation.

Tango utilizes a standard RS-232 serial port on the PC and connects to the UNIX computer via a modem or direct connection.

COSI

313 N. First St.
Ann Arbor, Michigan
48103
(313) 665-8778
Telex: 466568

Tango is a trademark of COSI, UNIX is a trademark of Bell Laboratories.



The PC-to-UNIX™Connection

Circle No. 267 on Inquiry Card

The Truth of the Matter is...

Prevail is a UNIX-based office automation and application development solution which can be shipped to you *today*. If you are looking for office automation software or need a fourth-generation language, look to Prevail—an A.T.&T. co-labeled product.

Prevail has seven components which will meet your needs.

- Word Processing
- Spreadsheet
- Database Management System
- Window Manager and User Interface
- Report Writer
- Applications Development Language
- Telecommunications

Prevail is available on AT&T 3B series, AT&T Unix PC Model 7300, NCR Tower, DEC VAX and MicroVax II series, Sun Microsystems computers, and Masscomp computers.

Inspiration Systems, Inc.
400 Cummings Park, Suite 4300
Woburn, MA 01801
617 938-1160

See Prevail on the IBM PC Booth No. 5124

COMDEX/Fall '85
November 20-24, 1985
Las Vegas Convention Center
Las Vegas, Nevada

Circle No. 297 on Inquiry Card

standably) protective of particular innovations to be announced later, or that such matters have yet to be decided, or both.

According to Laurence Brown, supervisor of UNIX Networking Systems Engineering at AT&T Bell Labs, "All that's been agreed to so far is that we will work together to ensure that there is a single UNIX standard that will both support current System V applications and will provide the networking services that traditionally have been offered on Berkeley-based systems." Now, System V, of course, already "supports current System V applications". Does this mean that the agreement essentially requires nothing more than the grafting of BSD networking facilities onto System V? Indications suggest that the process is somewhat more involved.

For its part, Sun's first step will be to add complete compatibility with AT&T's *System V Interface Definition* (SVID) to the Sun OS. While it is significant that another major UNIX vendor is making this move, it's not now considered news. Bill Joy, vice president for research and development at Sun, stated at the UniForum conference in Dallas last January that Sun would commit to SVID. Sun will port its Network File System (NFS) to System V, maintain 4.2BSD features and Sun enhancements, and incorporate 4.3 enhancements next Spring, but it's conceivable that Sun might have done these things even without the agreement with AT&T.

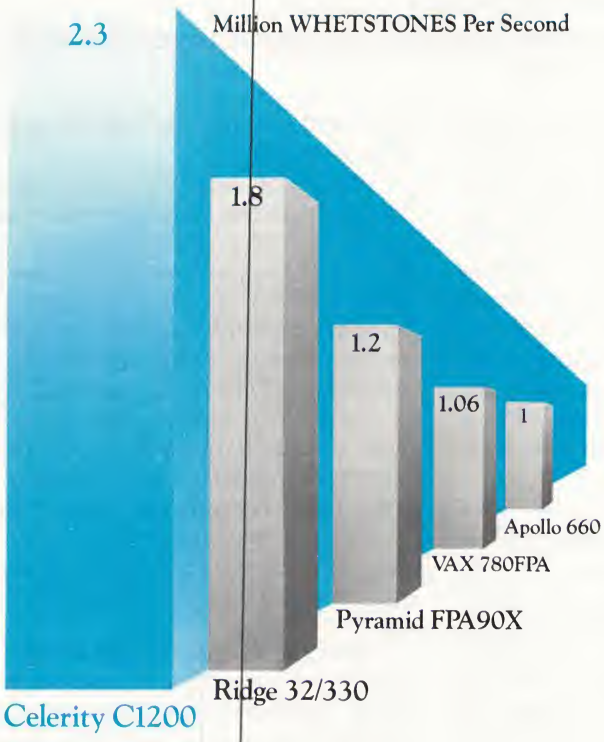
The new package is not to be a "dual" or "layered" port. In the Sun System V facility, system calls and other facilities required for System V are implemented as "native" extensions to the Sun OS kernel. A separate library is used for commands and utilities unique to System V.

Perhaps even more interesting than Sun's actions is the question of what AT&T will do. Since the focus of the agreement, as Brown stated, is on support for System V applications and the availability of networking services, and since AT&T is already very much engaged in the business of supporting System V, a major AT&T emphasis no doubt will be placed on networking. Brown observes: "The root of this agreement is that both companies feel applications are important—important to maintain compatibility for existing applications as our individual systems evolve; and that there is important new functionality coming in networking, and that it's important to define UNIX standards there. AT&T and Sun will work on those together as part of this agreement. . . . We saw networking as an area of potential divergence, and we'd like to bring everybody together there."

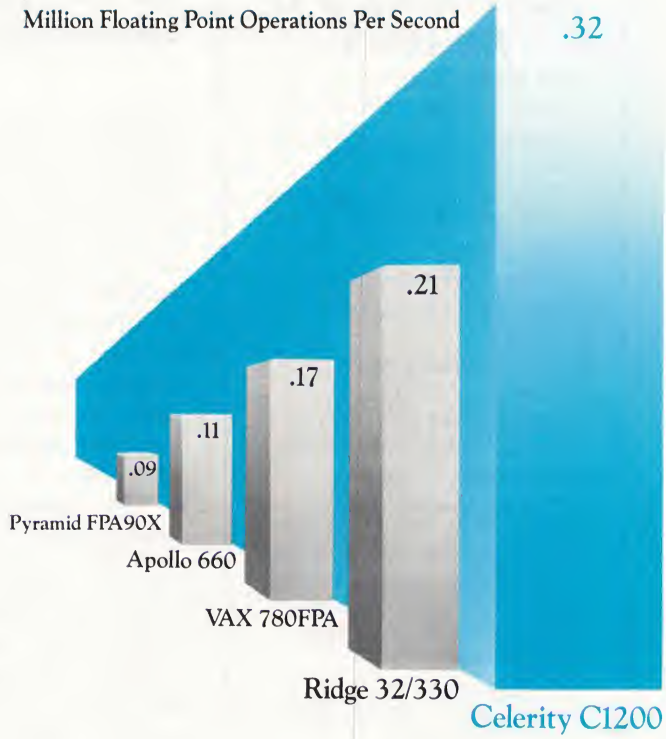
The fact that networking is a key issue in the agreement is public knowledge. What is not yet public are the specific facilities the companies will use in their joint networking scheme. "Now, the exact technology that's used to provide those additional [networking] services still needs to be worked out", Brown said, "and that isn't covered by the press release. . . . We need to agree on a common set of networking services that will be provided on all standard UNIX systems, and then any vendor, [in providing] upward compatibility for its customers, may extend beyond that and offer additional features on its systems."

This last remark leads to considerations of how AT&T's fundamental document, the SVID, may be altered by the Sun-AT&T agreement. Writing in the February, 1985, issue of UNIX REVIEW, Doug Kevorkian, supervisor of UNIX System Architecture and

Too Good To Be True?



WHETSTONE Benchmark
Single Precision



LINPACK Benchmark
Double Precision

Call Us On It.

The C1200 computational system continues to set new performance standards. Results from industry-accepted benchmarks highlight the C1200's performance when executing workloads characteristic of compute-intensive engineering and scientific applications. Similar performance results are achieved in a broad range of application environments: modeling, simulation, analysis, image processing.

The C1200 combines mainframe performance and features with unmatched local control to provide you the best price/performance value available today.

- Optimized native UNIX 4.2BSD
- 32-bit RISC-like architecture
- Up to 24 MBytes physical memory
- 4 Gigabytes virtual memory
- Multiple high speed buses
- Industry-standard graphics, compilers, networking and communications options
- Up to 32 users

The proof of performance is in the execution of your application. We promise performance. We deliver performance.

So Call Us On It.

CELERITY COMPUTING

Corporate Headquarters: 9692 Via Excelencia, San Diego, CA 92126 (619) 271-9940

Circle No. 266 on Inquiry Card

UNIX is a registered Trademark of AT&T Bell Laboratories. VAX is a registered Trademark of Digital Equipment Corporation.

CEEGEN-GKS GRAPHICS SOFTWARE in C for UNIX

- Full implementation of Level 2B GKS.
- Outputs, Inputs, Segments, Metafile.
- Full Simulation for Linetypes, Linewidths, Fill Areas, Hatching.
- Circles and Arcs, Ellipses and Elliptic Arcs, Bezier Curves.
- Ports Available on all Versions of UNIX.
- **CEEGEN-GKS** is Ported to Gould, Masscomp, Plexus, Honeywell, Cadmus, Heurikon, Codata, NBI, NEC APCIII, IBM-AT, Silicon Graphics, Pyramid, Tadpole Technology, Apollo, AT&T 3B2, AT&T 6300, DEC VAX 11/750, 11/780 (4.2, 5.2), NCR Tower.
- **CEEGEN-GMS** GRAPHIC MODELING SYSTEM: An Interactive Object-Oriented Modeling Product for Developers of GKS Applications. **CEEGEN-GMS** and **GKS** Provide the Richest Development Environment Available on UNIX Systems.
- Extensive List of Peripheral Device Drivers Including Tektronix 4010, 4014, 4105, 4109, HPGL Plotters, Houston Instruments, Digitizers, Dot Matrix Printers and Graphics CRT Controllers.
- **END USER, OEM, DISTRIBUTOR DISCOUNTS AVAILABLE.**

CEEGEN

CEEGEN CORPORATION
20 S. Santa Cruz Avenue, Suite 102
Los Gatos, CA 95030
(408) 354-8841
TLX 287561 mlbx ur

EAST COAST:
John Redding & Associates
(617) 263-8206
UNITED KINGDOM:
Tadpole Technology PLC
044 (0223) 861112

UNIX is a trademark of Bell Labs.
CEEGEN-GKS is a trademark of Ceegen Corp.

Circle No. 298 on Inquiry Card

Operating System Engineering at AT&T Bell Labs, stated, "In defining the relationship between System V and application programs, the SVID describes a minimum set of system calls and library routines that should be common to all operating systems based on System V. The remaining commands and utilities have been grouped into a logical series of optional extensions to the base definition."

The intent of the Sun-AT&T agreement is to incorporate BSD-derivative networking features into the SVID. Does this then mean that AT&T will adopt Sun's NFS? Sun's Bill Joy responded, "What has been announced so far is that [Sun] will supply an NFS for System V, and that NFS will be supportable under the AT&T networking scheme; in other words, whatever scheme AT&T has for supporting distributed file systems will support NFS. There hasn't been any announcement [yet] as to what AT&T's networking options for its customers will be." That is, Sun may or may not be sure how, but, as Joy added, "Half the code in our system is networking, so that has to get worked into the common framework somehow."

In seeking to determine what points are significant in the announcement of this agreement, representatives from all sides focus on the pivotal role of the SVID. This will determine how UNIX appears to the end user and the application; the technical manipulations that go on behind the interface are of secondary importance when one speaks of standards. Bernard Lacroute, executive vice president and general manager of Sun's workstation division, emphasized this point: "Of first and foremost importance is that, at the application level, a System V application or a 4.2 application can run without knowing whether or not it's System V or 4.2."

What of significance then comes from the announcement of the Sun-AT&T agreement? First, Sun will support the SVID. Second, System V will continue to support current System V applications, while being modified to provide BSD-derived networking services, the specifics of which will be announced later as the Sun-AT&T relationship matures. The news, then, is not that "The Standard Is Here", but rather that "another step in the continuing evolution of the standard is here".

There is a third point of substance, or perhaps it should be said, "potential substance". A popular computer industry perception holds that UNIX cannot be a standard because so many versions of it exist. AT&T, however, has a different perspective—one that claims the SVID is *the* standard UNIX base from which other vendors can add features. These features may give each version a different flavor, but the UNIX system at the base will remain standard. If this Sun-AT&T agreement contributes to the industry's adoption of AT&T's perspective, it will be substantive for that alone. "That is certainly our intention", said Bob Mitze, the department head of UNIX Computer System Development at AT&T Bell Labs. "That is our expectation—that. . .people will find that by writing to the SVID they can write portable programs that will move from machine to machine. We expect we will be able to solidify the standard to the point where we won't find ourselves with [the] perception [that the various UNIX versions are too disparate to be one standard]. Most programs turn out to be fairly easy to port. . . But the perception is nonetheless quite important, because that has a lot

CREATE
LASTING IMPRESSIONS

HIEROGLYPH™
UNIVERSAL REPORT PRODUCTION SYSTEM™

INTEGRATED TEXT/DATA/ADVANCED GRAPHICS SOFTWARE

A "report" is information presented in organized form—typically a printed document. If you are a professional whose work involves preparing reports HIEROGLYPH™ is for you. HIEROGLYPH is designed to meet the needs of technical and office report preparation and production where the combination of text, data and advanced graphics are essential elements. Whether you are an engineer, scientist architect, manager of business reports, a writer or graphic artist, HIEROGLYPH integrates all information in your UNIX Universe. HIEROGLYPH combines all the elements necessary to generate the final composition. You have the option of camera-ready copy or multiple copies in Color or Black and White. HIEROGLYPH uses simple, single-stroke commands. Both text and commands can be in English or several foreign languages. The user manual is written for three levels: New, Experienced and Expert. Regardless, you can be doing productive work the first day. HIEROGLYPH software incorporates Text processing, Document Aids, Document Filing, Graphics, Data Handling and Production Tools. Together they comprise the most productive system for creating and producing superior reports—documents that make lasting impressions.

PRESCIENCE

HIEROGLYPH is the product of Prescience, Inc. (pronounced pres-ce-ence) 820 Bay Avenue, Suite 300, Capitola, California 95010 (408) 462-6567.

NEW CORPORATE HEADQUARTERS
1825 SOUTH GRANT STREET • SUITE 510 • SAN MATEO, CA 94402 • (415) 573-1507

Circle No. 262 on Inquiry Card

to do with how many people are going to write software. . . . The market frequently seems to be based on perception."

ENCORE TAKES A BOW

A much-anticipated official announcement from Encore Computer Corp. has finally come to pass. Three new product lines and a version of UNIX are available as of this month: a family of general-purpose superminis; three models of interactive workstations; two models of a network communication computer (the "Annex"); and UMAX, yet another UNIX flavor.

The Multimax is designed to permit up to 20 main processors to share a common memory. Configurations cover a broad range of capabilities: performance that spans 1.5 to 15 MIPS; memory capacity ranging from 4 to 32 MB; and systems containing from one to 10 I/O channels. System prices begin at $112,000 for a dual processor (1.5 MIPS) system with 4 MB of shared memory, one I/O channel, one 515 MB disk drive, one 6250 bpi half-inch tape drive, and a workstation display or console printer. A large system with the same peripherals configured for parallel processing applications, with 20 processors (15 MIPS) and 32 MB of memory, is priced at $340,000. The Multimax superminis are aimed at the general-purpose computer market, and so compete with DEC VAXen and the Data General and Prime machines in this range.

Although the two low-end models in Encore's HostStation line of workstations—the 100 and 110—are single-processor machines, they are upgradable to the top-of-the-line 550, a desktop box with two 32-bit processors and a base package including high resolution (1056 by 864) 19-inch monochrome display, 1 MB of memory, 41 MB of internal hard disk storage (expandable to over 370 MB), three RS-232 ports; and a $14,000 price tag.

The Multimax family runs under UMAX 4.2, Encore's version of UNIX offering the full functionality of 4.2BSD. UMAX also offers parallel and distributed processing extensions, using thousands of hardware and software locks to protect individual elements within system tables. The system features "multithreading", a design providing simultaneous ac-

"Now we can build multi-user applications with a relational database—without the time and expense of programming.

THAT'S PROGRESS!"

Steve Stone, S.B. Stone & Company, Cleveland, OH

"We have been a major supplier of custom applications in the Cleveland market for seven years. We switched to PROGRESS,™ cut development times in half and are now delivering solutions to our users at lower cost, faster and more easily!"

PROGRESS is the only product that lets you build applications completely in a high-level fourth-generation environment. Its English language syntax increases productivity 10 to 40 times over COBOL, BASIC, and C. What's more, PROGRESS is a relational database with crashproof recovery, high-performance multi-user capability on large databases, and portability across UNIX,™ XENIX,™ and MS-DOS.™

If you want to save money and time on application development, call Data Language Corporation at 617-663-5000 and ask about PROGRESS.

DATA LANGUAGE CORPORATION
47 Manning Road, Billerica, MA 01821 617-663-5000

PROGRESS is a trademark of Data Language Corporation, developers of advanced software technology for business and industry. UNIX is a trademark of AT&T Bell Laboratories. MS-DOS and XENIX are trademarks of Microsoft Corporation.

Circle No. 289 on Inquiry Card

cess of system resources for multiple processors. (A single shared copy of UMAX resides in the Multimax shared memory, which facilitates multithreading. A uniprocessor version of UMAX operates the HostStation 550.) Multithreading is accomplished by multiprocessing primitives based on memory locks that provide access by multiple processors to operating system resources.

UNIX EXPO: BIG APPLE STAR

It appears Manhattan was twice taken by storm in September. Hurricane Gloria did her work the week of the 23rd, one week after the second annual UNIX Operating System Exposition and Conference, and while Gloria gratefully did not live up to dire predictions, UNIX Expo seems to have met its objectives and then some.

Held this year at the New York Hilton in Rockefeller Center, UNIX Expo is a business-oriented show, seeking not only to bring UNIX people together, but to assist UNIX companies in contacting potential customers: small companies, DP/MIS personnel in larger companies, VARs—any people or organizations considering the purchase of UNIX systems. Don Berey, account executive of show sponsor National Expositions Co., Inc., said those who came saw what they were hoping for: the event boasted 120 exhibitors and 10,460 attendees. The various conferences (four tracks covering UNIX and Office Automation, UNIX in a Data Processing Environment, UNIX Business Solutions, and UNIX and PCs) "played to standing-room-only crowds", and the tutorials, designed and developed by AT&T specifically for the show, each operated with attendance "at or near capacity".

Berey also pointed out that a large number of exhibitors have already reserved space for next year's Expo, to be held again in New York City, this time at the new Jacob Javits Convention Center, October 20-22.

David Chandler is the Associate Editor of UNIX REVIEW. ■

UNIX SYSTEMS UTILITY SOFTWARE —SO YOU CAN GET ON WITH YOUR JOB.

For more information, call or write. (703) 734-9844

UBACKUP
BACKUP, RESTORE, AND MEDIA MANAGEMENT

USECURE
SYSTEM SECURITY MANAGEMENT

SPR
PRINT SPOOLING AND BATCH JOB SCHEDULING

SSL
FULL-SCREEN APPLICATION DEVELOPMENT

S-TELEX
TELEX COMMUNICATIONS MANAGEMENT

SSE
FULL-SCREEN TEXT EDITOR

These products are available for most UNIX or UNIX-derivative operating systems, including System V, 4.2 BSD, 4.1 BSD, Xenix, Version 7, System III, Uniplus, and others.

UNIX is a trademark of AT&T Bell Laboratories.

UNITECH SOFTWARE INC. 8330 OLD COURTHOUSE RD. SUITE 800 VIENNA, VIRGINIA 22180

Circle No. 300 on Inquiry Card

THE HUMAN FACTOR

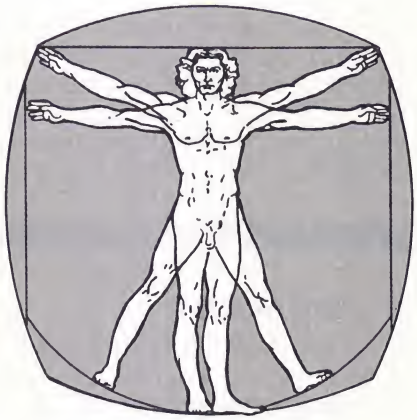
Of megaflops and multiprocessors

by Richard Morin

As noted in a previous column (January, 1985), scientists tend to have voracious appetites for computing power. This, along with their tolerance for new and unusual ideas, makes them good prospects for exploratory computer architectures. Consequently, many firms with unusual hardware designs select the scientific marketplace as their first target. Unfortunately, this has often consigned scientific users to peculiar and even barbaric excuses for operating systems. The scientists, needing lots of megaflops, haven't been able to be choosy.

A new day has dawned, however, and UNIX is coming to the rescue. Simply by being available, adaptable, and competently designed, it has become the operating system of choice for the current breed of offbeat scientific number crunchers. The fact that it has a significant following of users and vendors doesn't hurt either. Manufacturers are freed to produce just number crunchers, knowing that a wide range of workstations and other supporting components will be available from other vendors. Consequently, we see a host of vector processors, multiprocessors, RISCs (reduced instruction set computer), and other machines showing up at UNIX trade shows.

The fight isn't over, of course,



and a number of non-UNIX machines still are being developed. Some of these come from old-line manufacturers whose existing operating systems are quite satisfactory, at least to their current customers. Others, such as dataflow machines, reduction machines, and inference engines, are so peculiar as to make traditional operating systems such as UNIX entirely unsuitable. Still, the fact a large number of vendors have chosen to base all or part of their new ventures on UNIX is suggestive of a strong trend.

Before beginning our survey, a few words of warning may be in order. First, different architectures are optimized for different purposes, and a given machine may be entirely unsuitable for a given purpose, despite glowing performance figures. A vector machine that performs very well on large array calculations may be very poor at monte carlo analysis. Second, benchmark figures are always somewhat suspect, and published performance ratings are often chosen to favor a vendor's product. Thus, the figures offered here are more indicative than definitive. Finally, if real money is to be spent, a purchaser is well advised to investigate the track records of the models and vendors in question. Buying a low serial number product can occasionally be an all too interesting experience.

THE HIGH END

It's lonely at the top. Only a few companies are involved in the supercomputer game, and their customers—if few—are wealthy. Addressing hundreds of megabytes of RAM, and performing hundreds of millions of instructions per second, these machines are very powerful indeed. Some of the traditional players are still around, but a number of new companies have also arrived on the scene.

Cray Research (with headquarters in Mendota Heights, MN), inspired by hardware guru Seymour Cray, is the premier American producer of scientific supercomputers. Cray's 64-bit machines, optimized for fast floating point calculations and array ma-

Documentation and Software from Customer Information Center World Headquarters for AT&T Documentation

AT&T 3B Computer Manuals

Description	Select Code	Price
3B2 PC INTERFACE GUIDE 999-801-020IS	981-020	$24.00
PC 6300 PC INTERFACE GUIDE 999-801-021IS	981-021	$15.00
MF COBOL LANGUAGE REFERENCE MANUAL 999-802-00315	982-003	$35.00
MF COBOL LEVEL II OPERATING GUIDE 999-802-004IS	982-004	$35.00
RM/COBOL LANGUAGE REFERENCE MANUAL 999-802-020IS	982-020	$40.00
RM/COBOL USER'S GUIDE 999-802-021IS	982-021	$20.00
RM/COBOL RUNTIME GUIDE 999-802-022IS	982-022	$10.00
dBASE II USER'S GUIDE 999-803-000IS	983-000	$25.00
dBASE II REFERENCE MANUAL 999-803-001IS	983-001	$25.00
INGRESS SYSTEM OVERVIEW 999-803-002IS	983-002	$20.00
INGRESS QUERY-BY-FORMS GUIDE 999-803-003IS	983-003	$20.00
INGRESS REPORT-BY-FORMS GUIDE 999-803-004IS	983-004	$20.00
INGRESS REFERENCE MANUAL 999-803-005IS	983-005	$20.00
INGRESS VISUAL FORMS EDITOR USER'S GD 999-803-006IS	983-006	$20.00
INGRESS REPORT WRITER REFERENCE MNL 999-803-007IS	983-007	$20.00
INGRESS EQUEL/C PROGRAMMER'S GUIDE 999-803-008IS	983-008	$20.00
INGRESS SELF-INSTRUCTION GUIDE 999-803-009IS	983-009	$20.00
INGRESS ADMINISTRATOR'S GUIDE 999-803-010IS	983-010	$20.00
INFORMIX MANUAL 999-803-015IS	983-015	$45.00
FILE-IT! MANUAL 999-803-016IS	983-016	$35.00
C-ISAM MANUAL 999-803-017IS	983-017	$35.00
MULTIPLAN MANUAL 999-804-000IS	984-000	$35.00
BACS INSTALLATION GUIDE 999-806-024IS	986-024	$10.00
BACS ORDER INVENTORY MANUAL 999-806-025IS	986-025	$40.00
BACS PAYROLL MANUAL 999-806-026IS	986-026	$40.00
BACS ACCOUNTS PAYABLE MANUAL 999-806-027IS	986-027	$40.00
BACS ACCOUNTS RECEIVABLE MANUAL 999-806-028IS	986-028	$40.00
BACS GENERAL LEDGER MANUAL 999-806-029IS	986-029	$40.00
BACS WORKSHEETS 999-806-030IS	986-030	$5.00

Special Package Price

INGRESS PACKAGE OF 8 ITEMS (983-003 thru 983-010)	999-900	$120.00
BACS PACKAGE OF 5 ITEMS (986-025 thru 986-029)	999-901	$180.00

AT&T PC 6300 Documentation

REFERENCE MANUAL	637-400	$65.00
SERVICE MANUAL	637-800	$150.00
SYSTEM PROGRAMMER'S GUIDE	982-200	$65.00

AT&T UNIX PC 7300 Documentation

SERVICE MANUAL	962-030	$150.00
USER'S MANUAL	981-312	$65.00
PROGRAMMER'S GUIDE	981-313	$65.00

Special While Quantities Last

AT&T PC6300 System/Programming Software

		ORIGINAL PRICE	SPECIAL PRICE
DRI CBASIC COMPILER	021-105	$600	$299
DRI C	021-107	$350	$199
DRI PL/1	021-108	$750	$389
DRI PASCAL/MTt	021-109	$600	$299

CATALOG

THE UNIX SYSTEM V SOFTWARE CATALOG (Fall 1984 Issue)	307-125	$19.95	$13.95

ORDER TOLL FREE
1-800-432-6600
OPERATOR 363

AT&T
The right choice.

AT&T's Customer Information Center, Marketing Dept. 2855 N. Franklin Rd. Indianapolis, In. 46219

nipulations, are the standard of comparison for scientific number crunchers. A Cray X-MP, for example, can do about 250 megaflops (250,000,000 floating operations per second), for a mere $5 million. The Cray 2 is reputed to be faster still. And, naturally, Seymour is hard at work on the Cray 3. But what about software? COS, similar to CDC's NOS, has been Cray's historical proprietary operating system, but that picture has changed. Cray Research is using UNIX System V on the Cray 2, and says that it will port UNIX to the other models in the near future.

Some Japanese firms (Fujitsu, Hitachi, NEC) have produced very respectable supercomputers. A lack of software, among other things, has kept these machines from being distributed effectively outside of Japan. This is in the process of changing, however, and UNIX is playing a large role. All of these vendors have announced computers that run UNIX. In addition, the powerful Japanese Ministry of International Trade and Industry (MITI) has opted for UNIX as its primary standard. It is thus only a matter of time before UNIX-based supercomputers begin to arrive from Japan.

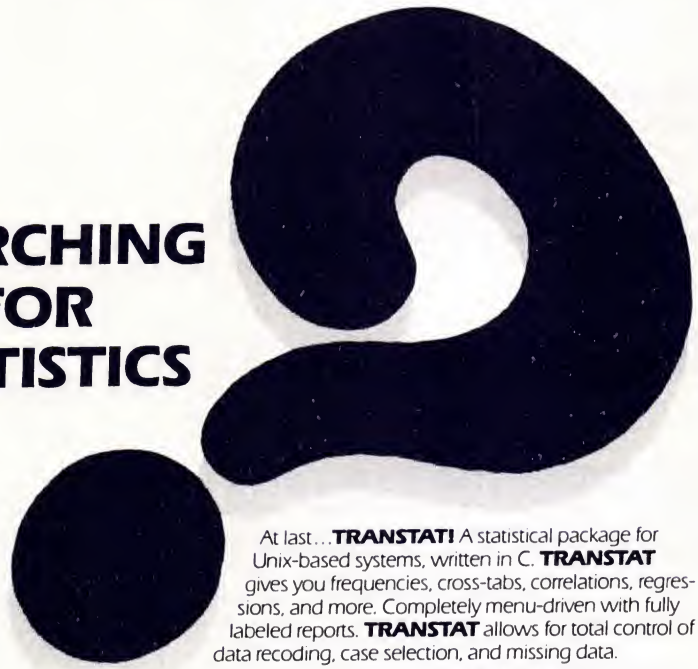
Denelcor (Aurora, CO) has not yet produced a machine that can take on a Cray, but it expects to do so in the near future. Currently, the firm produces only the HEP1 system, composed of up to eight processors, each of which can do 16 MIPS. Previously plagued by a lack of good support software, Denelcor has recently announced the introduction of a real-time, parallel processing version of UNIX for the HEP1. The HEP2, now being prototyped, is expected to be capable of 12K MIPS, putting it firmly in the supercomputer league.

ETA Systems (St. Paul, MN), a CDC spinoff, is scheduled to deliver its first ETA-10 UNIX-based, vector multiprocessor in late 1986. Delivering performance in the range of 10 gigaflops, the system will be able to support eight 64-bit vector processors, each with up to 32 MB of memory. In addition, the ETA-10 can have up to 2 GB of shared memory.

Though commercial supercomputers are generally not well optimized for scientific tasks, their powerful processing and I/O capabilities can occasionally be very useful. With the addition of attached array processors such as those made by Floating Point Systems (Beaverton, OR), a traditional commercial mainframe such as an IBM 3084 can easily qualify for scientific supercomputer status. IBM's interest in UNIX has been tepid to date, however, and its future directions are quite unclear. Still, IBM has (grudgingly) announced support for UNIX on its mainframe computers. Amdahl (Sunnyvale, CA) is also a name to be reckoned with in the commercial supercomputer field, and it has been a UNIX advocate for some years now.

Finally, any number of supercomputer designs are always brewing in assorted laboratories and universities. Many of these will never be built, and most will be of only academic interest. Still, it is this ferment that has produced many of today's hot machines, and it will no doubt

SEARCHING FOR STATISTICS



At last...**TRANSTAT!** A statistical package for Unix-based systems, written in C. **TRANSTAT** gives you frequencies, cross-tabs, correlations, regressions, and more. Completely menu-driven with fully labeled reports. **TRANSTAT** allows for total control of data recoding, case selection, and missing data.

Call for more information on **TRANSTAT** and Unix hardware, software, consulting and training.

BASIS

SPECIALISTS IN UNIX COMPUTING

1700 Shattuck Avenue Berkeley, California 94709 415 841 1800

Dealer inquires are welcomed.

TRANSTAT is a registered trademark of BASIS. Unix is a trademark of AT&T Bell Laboratories.

Circle No. 296 on Inquiry Card

YOU CHOOSE:

	MLINK	CU/UUCP
Terminal Emulation Mode		
Menu-driven Interface	Yes	
Expert/brief Command Mode	Yes	Yes
Extensive Help Facility	Yes	
Directory-based Autodialing	Yes	
Automatic Logon	Yes	Yes
Programmable Function Keys	Yes	
Multiple Modem Support	Yes	Yes
File Transfer Mode		
Error Checking Protocol	Yes	Yes
Wildcard File Transfers	Yes	Yes
File Transfer Lists	Yes	Yes
XMODEM Protocol Support	Yes	
Compatible with Non-Unix Systems	Yes	
Command Language		
Conditional Instructions	Yes	
User Variables	Yes	
Labels	Yes	
Fast Interpreted Object Code	Yes	
Program Run	Yes	
Subroutines	Yes	
Arithmetic and String Instructions	Yes	
Debugger	Yes	
Miscellaneous		
Electronic Mail	Yes	Yes
Unattended Scheduling	Yes	Yes
Expandable Interface	Yes	
CP/M, MS/DOS Versions Available	Yes	

MLINK™

The choice is easy. Our MLINK Data Communications System is the most powerful and flexible telecommunications software you can buy for your Unix™ system. And it's easy to use. MLINK comes complete with all of the features listed above, a clear and comprehensive 275-page manual, and 21 applications scripts which show you how our unique script language satisfies the most demanding requirements.

Unix System V	BSD 4.2	MS-DOS
Unix System III	Xenix	CP/M
Unix Version 7	VM/CMS	and more...

Choose the best. Choose MLINK.

Altos	Data General	IBM
Arrete	DEC	Onyx
AT&T	Kaypro	Plexus
Compaq	Honeywell	and more...

MLINK is ideal for VARs and application builders. Please call or write for information.

CMI Corporate Microsystems, Inc. P.O. Box 277, Etna, NH 03750 (603) 448-5193

MLINK is a trademark of Corporate Microsystems, Inc. Unix is a trademark of AT&T Bell Laboratories. IBM is a registered trademark of IBM Corp. MS-DOS and Xenix are trademarks of Microsoft Corp. CP/M is a registered trademark of Digital Research.

Circle No. 254 on Inquiry Card

continue to be a fertile source of new computer architectures. The ACM SIGARCH newsletters and conference proceedings contain many interesting descriptions of novel theoretical, experimental, and even commercially produced architectures. *Electronics* magazine is also a very good source for information on new commercially produced machines and interesting hardware trends.

These are exciting times for hardware junkies, and UNIX continues in its role as a distributed laboratory for computer science research.

CRAYETTES

It occasionally happens that one's processing requirements are not matched by a budget allowing the purchase of a multi-million dollar number cruncher. This could happen to anyone, but fortunately there are several vendors who are quite eager to help. The machines they produce, known as Crayettes, typically cost less than a megabuck, but provide as much as a quarter of the power of a Cray. Many of these machines are augmented by an assortment of vectorizing compilers and other software aids.

Several Crayette producers are making full vector processors. Two such machines, aimed directly at Cray owners, are produced by American Super Computer and Scientific Computer Systems. These companies have chosen to maintain binary compatibility with Cray 1 processors, and are even porting Cray's COS. This strategy may be short-lived, however, in light of Cray's move to UNIX. A number of other vendors have decided to go with UNIX, occasionally assisted by an underlying parallel kernel.

Alliant Computer Systems (Acton, MA) makes a multiprocessor 4.2BSD system that supports up to 256 MB of real memory, 2 GB of virtual memory, and a mixture of computational and interactive processors. At its full configuration of eight 32-bit vector processing computational elements, the system can reach speeds of 94 megaflops and 35 MIPS. The Alliant Fortran compiler automatically detects opportunities for parallel execution, allowing the runtime environment to perform entire DO loop bodies on multiple processors. Special hardware and software allow the system to deal with dependencies of one iteration on another.

Convex Computer Corp. (Richardson, TX) produces the C-1 64-bit pipelined vector processing system, which runs an operating system based on 4.2BSD. The C-1 is able to do 60 megaflops and handle up to 128 MB of memory, while maintaining VAX/VMS Fortran compatibility. An interesting technique known as "disk striping" is now being used by Convex. With this technique, a set of disk drives is treated as a single drive.

Great-looking TROFF output from low-cost laser printer!

■ Now! Full support for LaserJet+ ■

For several years, Textware has been licensing TPLUS† software to process the output of TROFF and DITROFF for a wide variety of phototypesetters, laser printers, etc. Now, with TPLUS driving the LaserJet+, we have *again* set a new standard for price/performance. By adding our *Graphics Option*, with DWB‡, you have the total solution to your document production requirements.

Many organizations are now getting maximum benefit from the HP LaserJet, using our TPLUS/LJ software. The low-cost LaserJet is a remarkable value on its own—8 page per minute output speed, 300 dot per inch resolution, and typesetter-quality fonts. TPLUS gives you access to all this *and more* from your own system. We support all the characters and accents needed by TROFF and EQN; in addition, special characters (☺; logos too) can be supplied or generated to meet specific requirements. Our precise handling of rules and boxes allows you to take full advantage of TBL for forms, charts, etc.

EQN examples	
$\lim_{x \to \pi/2} (\tan x)^{\sin 2x} = 1$	
$\frac{\alpha+\beta}{\sin(x)}$	$\prod_{k \geq 1} e^{S_k z^k / k}$

While even LaserJet output is not in the same class as the best phototype, it is certainly well suited to documentation and a broad range of other applications. When you do have a need for phototypeset images, TPLUS and the LaserJet will save you time and money. *Preview mode* lets you proof *all* aspects of your documents conveniently, in-house, before sending out for phototypesetting (from our UNI•TEXT service). Cross-device proofing is a standard feature of TPLUS.

The HP LaserJet printer is not only inexpensive—it is an exceptional value! Want proof? This entire ad was set in position using TPLUS on the LaserJet!

† TPLUS is a trademark of Textware Intl. ‡ Documenter's Workbench is a trademark of AT&T

For further information, please write or call.

Also available for:

- AM 5810/5900 & 6400, APS 5 & μ5, CG 8400 & 8600, Mergenthaler 202
- Xerox 4045, 2700/3700 & 8700/9700
- BBN, Sun, 5620 & 'PC' CRTs
- Diablo, Qume & NEC LQPs
- C Itoh & Epson dot-matrix

TEXTWARE INTERNATIONAL
PO Box 14 Harvard Square Cambridge, MA 02238
Telephone: (617) UNI-TEXT

Circle No. 293 on Inquiry Card

NEW VERSION!

Come to TERM with your Unix/Xenix communications problems.

TERM - More Powerful. Easier To Use.

Compare These Special Features:

- Easy to remember mnemonic commands
- Online user's manual for instant help
- Menu driven interface
- Fast - 9600 baud file transfers
- Self installing
- Powerful scripting language with variables
- Wildcard file send/receive capability
- Automatic error-checking and re-transmission
- Xon/Xoff, Etx/Ack, Line and character protocols for communications with non-TERM systems
- Xmodem protocol for remote bulletin boards
- Full/half duplex emulation modes
- Automatic login and logout
- Auto-dial, auto-redial, answer and hangup modem support
- Unlimited phone number directory for auto-dialing
- Unattended file transfers
- Remote maintenance capability
- Sample scripts included
- MS-DOS and CP/M versions available

TERM - Powerful Communications.

TERM - Unix/Xenix's most powerful communications program. TERM Communications Software provides a full-featured, programmable communications tool under the Unix/Xenix environment.

You'll appreciate TERM's ease of use, compatibility with a wide user base, and ability to talk to most other systems. TERM is both a smart terminal and file transfer program. It has extras you won't find in other Unix communications programs: On-line HELP, character translation, efficient error-checking protocols and file transfers for text and binary data.

TERM provides full modem control, an extensive script language, auto-login and logout functions, and can be run unattended for remote maintenance.

TERM is available **NOW** on the Altos 586, 2086, IBM AT, Tandy Model 16, 6000, AT&T 3B2, IBM PC/XT, and many others. Find out how easy it is to get your Unix, Xenix and MSDOS machines all talking together.

$295.00

TERM

COMMUNICATIONS SOFTWARE

Call or write for more information.

CENTURY SOFTWARE

9558 South Pinedale
Salt Lake City, Utah 84092
(801)943-8386

VISA / MC

Unix is a trademark of AT&T Bell Laboratories. MS-DOS and Xenix are trademarks of Microsoft Corp. CP/M is a registered trademark of Digital Research Inc.

Circle No. 251 on Inquiry Card

Concurrent writing allows data transfer rates to be multiplied, and the increased apparent size of the disk allows much larger files to be handled.

Other vendors have opted simply to produce fast multiprocessor scalar machines. The most aggressive design of this sort comes from Flexible Computer (Dallas, TX), whose FLEX/32 can combine up to 20,840 processor cards, currently based on the 32-bit NS32032 microprocessor. A key design factor, however, is the system's ability to integrate many different kinds of processing elements. Supporting real-time as well as number crunching applications, the system allows both hardware and software reconfiguration to be done while the system is running.

ELXSI (San Jose, CA), another multiprocessor vendor, has chosen instead to use small numbers of very powerful processors. The ELXSI product is perhaps more of a parallel mainframe than a mini-supercomputer. Its 300 MB-per-second bus supports up to twelve 64-bit processors, each of which is approximately equivalent in power to a DEC VAX 8600. The processors can share 200 MB of memory, to be quadrupled with the introduction of 256 kilobit RAM chips. A company spokesman notes that parallelization of code is often far easier than vectorization, and that such programs as SPICE are easily and efficiently run on ELXSI architecture.

In its newly announced iPSC system, Intel Scientific Computers (Beaverton, OR) has opted to use CalTech's hypercube architecture. In this design, 2^N processing nodes are used, with each node being able to communicate directly with N other nodes. The iPSC can be purchased in configurations of 32, 64, or 128 nodes, with each node containing an 80286 CPU, an 80287 FPU, and 512 KB of memory. The system is controlled by a UNIX-based "cube manager", which is responsible for resource management, user interface, and other support functions. At 25 to 100 MIPS and 2 to 8 megaflops, the iPSC is hardly a full supercomputer, but at $500,000, it does provide a relatively economical base for research into arbitrary multiprocessor topologies.

Literally dozens of vendors are producing multiprocessor or otherwise unusual UNIX systems. February's UniForum trade show in Anaheim will no doubt be full of such vendors hawking their wares, with the offbeat systems standing next to the YAWN (Yet Another Workstation or Network) products. These are exciting times for hardware junkies, and UNIX continues in its role as a distributed laboratory for computer science research.

Mail for Mr. Morin can be addressed to Canta Forda Computer Lab, PO Box 1488, Pacifica, CA 94044.

Richard Morin is an independent computer consultant specializing in the design, development, and documentation of software for engineering, scientific, and operating systems applications. He operates Canta Forda Computer Lab in Pacifica, CA. ■

MORNING STAR TECHNOLOGIES

UNIX* COMMUNICATIONS

X.25 • HASP • SNA3270 • SNA3770

Drop-in communication systems for MULTIBUS* based computers. Offload the CPU intensive process of communication with the HORIZON™ Series of boards from MORNING STAR. Complete systems include your choice of hardware and software combinations to custom fit your data communication needs. Available for: **Sun Microsystems, Masscomp, Pyramid, Heurikon, Plexus, NCR Tower, Sperry 5000, Celerity and more.**

Call today for more information

800 - 558 - 7827

Morning Star Technologies, Inc.
1760 Zollinger Road, Columbus, Ohio 43221

In Ohio call (614) 451-1883 TWX - 510 - 600 - 3272

*UNIX is a Trademark of AT&T Bell Labs • MULTIBUS is a Trademark of Intel Corp.

Circle No. 292 on Inquiry Card

The Firebreathers continue on the cutting edge of high performance computers.

The most powerful line of computer systems made. Gould PowerNodes™ and CONCEPT/32s®

Any way you slice it they beat the VAX.™

Our mainframe PN9000 and CONCEPT 32/97 are up to twice as fast as the VAX 8600.

And even though the mid-range PN6000 and CONCEPT 32/67 are 30-50% smaller than the VAX 11/780, they're still up to three times more powerful.

More power for a slice of the price.

Despite their superior power, our mid-range models cost 40% less than the VAX 11/780. Our mainframes cost about 30% less than the new VAX 8600. The bottom line is more power for less money.

Operating environments that are a cut above the rest.

There's also a choice of system software to consider. Gould's unique UTX/32® is the first operating system to combine UNIX* System V with Berkeley BSD 4.2. So it allows you to access virtually any command format you want whenever you want.

And in real-time environments, Gould's MPX/32™ operating system offers performance that's unmatched in the industry, as well.

Delivery that's right on the mark.

Unlike the VAX 8600, that has up to a 12 month wait for delivery, when you order either a Gould PowerNode or a CONCEPT/32 system, they'll be shipped within 90 days ARO.

You can also be sure with Gould you're getting a computer that's backed by years of experience – the kind of experience we used to develop the first 32-bit real-time computer.

If you need more information or just have a few questions, give us a call at 1-800-327-9716.

See for yourself why VAX no longer cuts it. Go with a Gould computer and ax the VAX.

CONCEPT/32 and UTX/32 are registered trademarks and PowerNode and MPX/32 are trademarks of Gould Inc. VAX is a trademark of Digital Equipment Corp. UNIX is a trademark of AT&T Bell Labs.

GOULD
Electronics

Only Gould computers have a big enough edge to ax the VAX.

Circle No. 268 on Inquiry Card

THE FINAL FRONTIER

UNIX and scientific applications: symbiosis or antithesis?

by Joseph S. Sventek

The popularity UNIX enjoys in many segments of the computing community is hardly a secret. Historically, it has been the operating system of choice on minicomputers in academic circles. Its recent availability on supermicro computers also has made it an attractive system for the business community. Even the home computer market has been affected by Xenix, PC/IX, and various UNIX work-alikes.

There is, however, one major segment of the computational fraternity that has received UNIX with something less than enthusiasm—the scientific community. This is not to imply that UNIX cannot be applied to scientific problems—the remaining articles in this issue provide evidence to the contrary. Even so, there are some legitimate reasons for the reticence scientists have shown in adopting UNIX. This article explores those reasons and offers a prognosis for the future success of UNIX systems in this marketplace.

There is one major segment of the computational fraternity that has received UNIX with something less than enthusiasm—the scientific community.

A TAXONOMY OF SCIENTIFIC APPLICATIONS

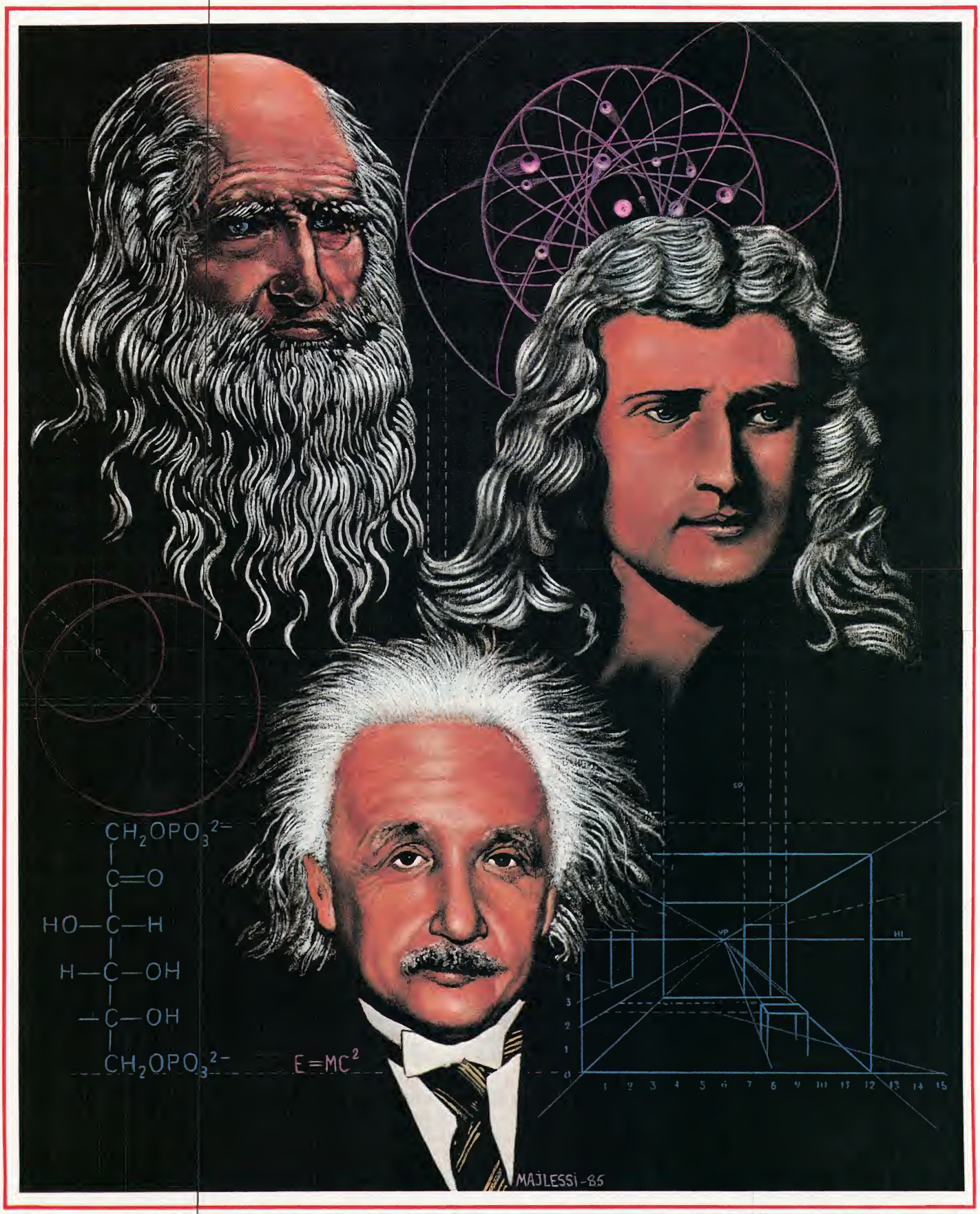
The first major category of scientific applications might best be described as "computationally intensive". These applications are of two general types: numerical simulations of physical phenomena and analysis of experimental data. Applications of either type make huge demands on a CPU's ability to perform floating point operations (FLOPs). The operating system features with the biggest impact on the execution of these applications are unlimited process address space and high-level compilers capable of generating efficient floating point object code.

The second major category of scientific applications is made up of event-driven tasks, which can be partitioned into data acquisition systems and experimental control systems. Each application must be able to respond *quickly* to external events, where this quickness depends on the system being measured or controlled. The critical services provided by an operating system in support of this category include: 1) small (or bounded) interrupt latency, 2) a user-tailorable priority scheduler, and 3) non-blocking system services accessible to user processes.

One class of applications in particular represents a combination of the "computationally intensive" and "event-driven" requirements: computer graphics. Graphical summaries of simulated or analyzed data usually entail significant amounts of floating point computation, while real-time displays supporting event-driven applications represent additional peripherals for which the processing time must be bounded. Applications of this sort require that the operating system provide interactive display tools and standard graphics library calls for program invocation.

In summary, for a single operating system to support all major categories of scientific applica-

Illustration by Heda Majlessi


CH2OPO3 2−
C=O
HO—C—H
H—C—OH
—C—OH
CH2OPO3 2−
E=MC2
MAJLESSI-85

tions, it must provide the following facilities:

- unlimited process address space (virtual memory).
- efficient high-level language compilers.
- bounded interrupt latency.
- priority scheduling.
- user-mode asynchrony.
- standard graphics subroutine libraries.
- interactive graphics utilities.

A SHORT HISTORY OF SCIENTIFIC COMPUTING

Although we are primarily concerned with the relevance of UNIX to scientific computation, knowledge of the culture that has developed in scientific computing circles will help us more fully understand the situation. It is important to note that, after code breaking, computationally intensive scientific applications made the first major use of early computers. These early programs were written in machine language.

As scientific computation grew more commonplace, it became apparent that the low-level programming languages of the day (machine and assembly) were hampering productivity severely. In response to this, one of the first *high*-level languages, Fortran (FORmula TRANslation), was designed to permit scientists to program in a language closer to the algebraic formulas used in their initial derivations of problems. The tremendous improvement provided by Fortran quickly led to its adoption as the *lingua franca* of scientific computing in the early 1960s. Since most computational resources were quite scarce, Fortran compilers developed a reputation for generating very efficient object code.

One factor that often determines the envelope of experimental and theoretical science is the amount of computational power that can be brought to bear on the problems at hand. Other system considerations (like the command language, the program development environment, and the file system) are secondary to the number of FLOPs that can be performed. As a result, organizations with an insatiable appetite for FLOPs (such as the US Department of Energy laboratories) tended to order only bare hardware from *supercomputer* manufacturers in the early days of scientific computing. The system programming staffs of these organizations would then craft minimal batch or timesharing operating systems on top of this iron. The highest priority item during the development of these operating systems was always an optimizing Fortran compiler. Other aspects of the system were usually made compatible with previous systems—leading to virtual immortality for a number of primitive operating system interfaces.

UNIX, as it is commonly delivered, is not able to provide the facilities necessary to support event-driven applications.

As scientists started to become involved in event-driven applications, they naturally wanted to use a programming language with which they were familiar. As a result, local extensions to Fortran (both the language and its runtime library) were implemented to permit the language's use in these real-time systems. Such local extensions generally caused a decrease in the productivity of programmers because of the mobility of scientific researchers. The fact that these originators often were not available for later support of their software meant that others either had to live with the problems they found or had to re-write whole sections of code. To guard against this, several progressive standards were developed for the Fortran language (ANSI X3J3).

This all has served to make a good Fortran compiler essential to a scientific computer system. The compiler must accept programs written in standard Fortran and generate efficient object code. Most scientists don't care much about the rest of the system, opting for compatibility with the past whenever a choice is available.

STANDARD UNIX SUPPORT FOR SCIENTIFIC APPLICATIONS

UNIX provides many of the facilities necessary to support the computationally intensive class of scientific applications. In particular:

- With the introduction of 3BSD, virtual memory support in UNIX systems was established. Berkeley has continued to improve the virtual memory support in BSD releases, while many commercial vendors—AT&T included—have begun to offer their own support for virtual memory.
- A Fortran compiler (f77) is provided. While this compiler correctly processes standard Fortran, some of the design decisions in the construction of the compiler prevent it from generating efficient object code. (See

Continued to page 44

UNIX™ & 'C' TRAINING

Whether you're training 2000, 200, or two...you can select the most efficient and economical training solution for your unique environment.

VIDEO-BASED TRAINING for professionally produced, consistent training that is always available at your location.

INTERACTIVE VIDEODISC TRAINING, using state-of-the-art technology to dynamically tailor courses to the individual—from novice to expert programmer.

PUBLIC SEMINARS offered in major cities throughout the world: *UNIX Overview • UNIX Fundamentals for Non-Programmers • UNIX Fundamentals for Programmers • Shell as a Command Language • 'C' Language Programming • Shell Programming • Using Advanced UNIX Commands • UNIX Internals • UNIX Administration • Advanced 'C' Programming Workshop • Advanced 'C' Programming Under UNIX • Berkeley Fundamentals and 'csh' Shell.* **ON-SITE SEMINARS** for training customized to your system and to specific groups within your organization.

ASK FOR OUR 48-PAGE COURSE CATALOG AND CURRENT SEMINAR SCHEDULE, CALL (800) 323-UNIX or (312) 987-4082

Three factors make the Computer Technology Group the experts in UNIX and 'C' language training:

- Experience, through training thousands of students worldwide in live seminars, with thousands more using our video training at their locations.
- Extensive Curricula Supporting All UNIX Versions, creating a client base of manufacturers, software developers and end users.
- Quality of Instruction, with instructors and course developers who are experts in teaching UNIX and 'C', as well as in designing and implementing a variety of UNIX-based systems.

COMPUTER TECHNOLOGY GROUP

Telemedia, Inc.

310 S. Michigan Ave.
Chicago, IL 60604

The Leading Independent UNIX System Training Company

™UNIX is a trademark of AT&T Bell Laboratories.

Circle No. 247 on Inquiry Card

A RUN THROUGH THE MILL

Experiences with scientific data analysis using UNIX

by Robert Goff

Scientific computing is a very mixed bag. It seems that scientists are particularly imaginative when it comes to devising applications that find the weaknesses in an operating system or hardware configuration. The diversity in functionality required for even relatively simple scientific data analysis leads to system complexity and performance requirements almost unheard of in other major segments of the computing industry. It also raises the inevitable question: *is UNIX really up to it?*

A complete answer, of course, is not possible in the space allocated to this article, nor, for that matter, in this entire issue of UNIX REVIEW. Rather, what will be attempted here is a discussion of a few of the adversities awaiting the scientific system developer and a sampling of problems that UNIX has played a key role in solving.

CONVENTIONAL WISDOM

As in any other discipline, much of the "knowledge" surrounding the use of UNIX for particular types of applications comes in the form of *old wives' tales*. Most of these tales have some basis in fact or history, but none should be taken at face value without investigating the implications for the specific application at hand. Many of the restrictions that purportedly beset UNIX either only apply to a restricted set of problems or reflect deficiencies that already have been solved over the course of the operating system's evolution. The old *unstable file system* bugaboo is a classic example of a malady that no longer plagues UNIX.

Conventional wisdom says if you are trying to acquire even a moderate amount of data in real time, you shouldn't use UNIX. Everybody knows that UNIX does not function well when subjected to the high interrupt rates characteristic of this kind of machine activity. Either you won't be able to keep up with the incoming data or you'll have to assign the acquisition such a high priority that the machine will become sluggish, the clock will lose time, the disk will *blow revs*, and the crash rate will go up alarmingly. But

Illustration by Stephen G. Luker


IBM
IBM
Steve Luker
0·1·2·3·4·5·6·7·8·9

how much of this is really true? And what does it mean to your application?

At the heart of this consideration is the question: *what do you mean by real time?* A similar, equally perplexing issue is the need to come to grips with what a *development environment* is and how that differs from a *production environment.* The virtues of UNIX in a development environment are well known by now and no doubt account for a large component of the system's popularity in the research community. After all, the major cost component in most development projects is *people time*—a currency that well displays the value of UNIX. But does this mean that the system's value goes down as we get closer and closer to *freezing the code*?

In scientific data analysis, a similar contrast often is suggested between *off-line* and *research mode* (interactive) processing. Does it become nothing more than a question of volume, or is there something fundamentally different in the way these two types of computing are done? If a difference is suspected, is it borne of convention or necessity? Can UNIX support a high volume processing environment? Alas, there is no universal answer. The question that really needs to be asked is: *what is meant by "high volume"?*

REAL TIME AND UNIX

As a simple illustration, I will describe a data acquisition system now under development. The hardware is a MC68000 Multibus box running 4.2BSD. We intend to acquire approximately 20 channels of seismic data and digitize it at 20 samples per second to 16-bit accuracy. The sampling operation must be synchronized with coordinated universal time (UTC) to obtain an absolute sample timing uncertainty of less than 20 milliseconds. The application also must run in the presence of other processes that route the data to its final destination over a communications link.

0•1

It seems that scientists are particularly imaginative when it comes to devising applications that find the weaknesses in an operating system or hardware configuration.

In view of the relatively low data rate and the short time allotted for device driver development, we chose a simple, inexpensive multiplexed analog-to-digital converter board offering two major programming modes, both of which generate an interrupt-per-sample. In retrospect, it probably would have been wiser to spend about half again as much money on this module to get a board with some on-board buffering and a DMA bus interface. This was rejected, however, since all the boards that we found required some custom firmware development for an on-board micro. We simply chose not to devote that much time to this aspect of the development.

The device driver for the system took about two weeks to write and was designed to allow us to use the board in any of its programming modes with any configuration of inputs. We knew that at a rate of approximately 400 interrupts per second, problems were likely, so we paid particular attention to streamlining the interrupt handler in the hope that we might avoid burning system time at an inordinate rate. Of equal importance was the fact that we needed to maintain an overall system throughput that would allow us to keep up with all of the tasks associated with reformatting and transmitting the data—and would allow us to maintain a reasonable set of records on the state of the system's health. The upshot is that the system had to be designed so that it wouldn't monopolize critical system resources unnecessarily. This amounted to integrating acquisition tasks into the system in a way that was *polite* to other activities.

We next wrote some user-level code to read the data. This is where we really started to learn how the system would react. The first programming mode we tried, known as *random mode*, required that the interrupt handler supply a new channel and re-arm the converter trigger for each sample. We started experimenting with relatively low trigger rates before cranking up the speed to see if we could reach our goal of an aggregate conversion rate of 400 samples per second. The actual conversion process takes only about .4 milliseconds, so we could tolerate interrupt service delays of as much as 24.6 milliseconds before data overruns would occur.

We were somewhat disappointed to learn that even at a low 100 sample-per-second rate, we experienced an unacceptable number of data overruns—even when the machine was unloaded. We tried busy/wait loops in place of

kernel sleeps and used other methods that made the acquisition less polite but it quickly became clear that we were on the wrong track. Without placing the hardware interrupt priority of the board considerably higher than we wanted, we could see that this mechanism was not going to work.

The other programming mode that was available to us is called *scan mode*. Under this scheme, a trigger is used to initiate an entire scan that starts at some low channel number and proceeds to some higher channel number. The trigger begins the first conversion and upon completion, issues an interrupt. The act of reading the data initiates conversion on subsequent channels until the high channel is reached, at which time an end-of-scan is signaled and the board stops, waiting for the next trigger. Note that the sequence involves fielding just as many interrupts as random mode would require but allows the interrupts to be spread out in accordion fashion over the full time interval allotted for the scan.

In order to use scan mode, we knew we would have to give up the generality of being able to sample the channels in any order. We decided, though, that this sacrifice was of minor practical importance since the data had to be massaged before transmission if samples were to be rearranged with little penalty. Of more concern was the fact that scan mode would not allow us to control absolute sample timing as well. Conversions became a function of how quickly we could get around to servicing the interrupts for all the earlier channels in a scan. To solve this problem, some instrumentation was necessary to determine the length of the average scan. If we could get a good estimate of the constant portion of the delay between the trigger and conversion for each channel, we felt we could at least remove that component of the timing error.

2·3

The major cost component in most development projects is *people time*—a currency that well displays the value of UNIX.

Using scan mode, we found that we could speed up the aggregate conversion rate to more than 1000 samples per second in the presence of other processing before we experienced any data overruns. It should be added that the prototype system was connected to other machines at our facility via a local area network and had no disk of its own. A fairly severe test of the system came when we inadvertently left the **rwho** daemon running during one of our test runs. The **rwho** daemon presents an intermittent load to the system and makes network traffic, in particular, fluctuate wildly. These tests led us to another modification of our approach.

We had allowed the interrupt handler to place newly arriving samples into a buffer (allocated in the device driver) that could hold about 1 second of incoming data. We devised a simple wrap-around scheme for handling buffer overflows and a sequencing method to report them as they occurred. The user-level code we were using read one scan at a time and placed the data on disk files (across the network). Only when we started to add functionality did further problems arise.

It seemed we just couldn't make the program run fast enough (or get enough of the CPU) to keep up with the data. The application would run for a while and sooner or later produce a buffer overflow. I suggested that we instrument the code until we really understood where the bottleneck existed. A few hours later, one of the programmers on the project mentioned that he'd noticed **rwho** running while he was performing his tests, leading him to wonder if this might be the source of the trouble. Of course, his comment led us right to where the real problem had been all along—in the device driver's small buffer size. When we expanded this buffer to accommodate 8 seconds of data, all problems of this nature disappeared.

WHAT ARE THE LESSONS?

The two solutions illustrated above are not terribly surprising in and of themselves—in fact they are fairly obvious to everyday users of UNIX. What *is* surprising (or at least was to me) is the magnitude of the effect they portray. I never would have thought that we could squeeze a ten-fold sampling rate increase out of our system simply by easing the critical path problem presented by the interrupt service mechanism. Nor would I have dreamed that we would have to let our user-level code ignore the incoming data stream for anything close to 8 seconds during routine processing.

What this illustrates is that, compared to smaller, less functional operating systems, UNIX gives the appearance of a higher degree of asynchrony in its man-

agement of machine activities. On the face of it, this may seem to be a disadvantage in that system buffers need to be larger and critical paths may become more numerous and difficult to forecast. But, though critical path management may be more important, the facilities provided by the operating system for doing it are more numerous and general, and the help they provide to the system developer may result in better overall system throughput when the processing load is analyzed as a whole.

As the limitations on system throughput are explored, it's invariably found that one critical resource or another is in short supply. Under UNIX, there may be a few more procedure calls between you and the handling of an interrupt or the reading of some data—and this may affect maximum system throughput if the resource in short supply is CPU time—but UNIX also provides some tools for use in dealing with these problems.

Without extraordinary effort, anything approaching total utilization of all system resources is unlikely. So the task faced by the system developer is to offload processing from resources that are approaching saturation to those that are under-utilized. UNIX helps with these efforts.

In the final analysis, the resource that the system developer must manage most carefully is development time. After only two weeks of development, we had a device driver with only one minor deficiency—one that was easy to remedy. Further, the level of functionality and flexibility we were able to achieve in that time was significantly improved by the completeness and generality of the model on which UNIX device driver implementations are based. The availability of facilities such as a general-purpose kernel **sleep** mechanism can be invaluable when efficient use of critical resources becomes imperative.

Since this application does not tax most of the resources provided by our machine (the CPU in particular), it is clear that a significant increase in capacity should be possible. The job of implementing a device driver for a higher performance A/D system would not, in my estimation, be significantly harder or more time consuming than for the simple device we have used. Further, the impact of the acquisition on the remainder of a machine's processing load should, if anything, be more controllable since the device in question should have more *intelligence* and its features should be fairly accessible. Thus, custom firmware development aside, an attractive expansion path for the system seems to exist. In our case at least, this owes in part to our choice of UNIX to do the job.

4•5

Without extraordinary effort, anything approaching total utilization of all system resources is unlikely.

DIFFERENT GOALS, DIFFERENT APPROACH

To show how some of these issues stack up in the larger scheme of things, I will describe another data acquisition system I took part in developing. The main emphasis of this project was directed toward doing a small amount of processing on a large volume of data at a minimal hardware cost. To accomplish this, a PDP-11/23 system running RSX-11M was chosen for the development work, with the idea that we would run the production system under RT-11 once we were done.

The project involved acquiring 30 channels of data from a microwave telemetry link connected to a small array of seismic stations some 70 miles away. Each channel was to be sampled at 250 readings per second, making for an aggregate data rate of 7500 samples per second—one considerably higher than in the previous example. The only processing to perform, however, consisted of demultiplexing and time-stamping the data before passing buffer loads of it to another machine (a PDP-11/34), via DMA, for further processing.

Since no *off-the-shelf* interface was suitable for connection to the telemetry, we had to develop our own. We chose to build a fairly fancy device for offloading the demultiplexing from the 11/23 CPU. This DMA device only required that it be told from time to time (about once a second) the hunk of memory into which it needed to poke its data. It then signaled the CPU with an interrupt whenever a buffer filled and a new address was needed.

Since disposing of the data also was to be done via DMA, most of the heavy work could be handled by specialized hardware. The main tasks for the CPU consisted of reading time information from a UTC clock, attaching that information to buffers as they were filling, doling out buffer addresses as needed, and responding to operator requests for system monitoring information.

As it turns out, several things slowed the pace of our project, not the least of which was our initial unfamiliarity with the DEC oper-

ating system and hardware. We had decided early on that since the development of our telemetry interface would take a fair amount of time, we would have to undertake the software development in parallel. Initial development proceeded rather slowly and resulted in a system that was far from expandable either in capacity or functionality.

By the time the system finally gave in and showed signs of working, we had evolved to a standalone-system approach for the software. This is not to say that the operating system we had chosen was unsuitable or deficient, but simply that we found we were using less and less of its facilities as time went by. We finally decided that we could do without it altogether since the activities we were trying to manage were few in number and highly synchronous in nature. In doing so, we gained considerably tighter control over utilization of the resources at hand.

The definition of this system was characterized by fairly tight machine constraints, a narrow set of goals, and the fact that the desired result of our development was a *black box*—a prime candidate for cross-development under UNIX. After re-writing our early assembly language software in C (and purchasing a C compiler for the 11/23), we now have a system that can be modified easily and yet still boasts the high performance/cost ratio we were looking for.

WHAT ABOUT ANALYSIS?

Scientific data analysis is another class of problems for which UNIX has some unique solutions. Like people, analysis techniques come into the world, grow up, get old, and sooner or later die. During the early stages of this life cycle, a technique's developers will try to ascertain the applicability of their new analysis tool, so a fairly rough and unsophisticated implementation of the idea usually will suffice. If a scheme proves useful and is allowed to continue its growth toward maturity, it will be cloned many times and the copies that are sent out into society will be molded by their environment into tools directed at the specific needs of particular projects. As these techniques grow more sophisticated, some may be trained toward higher and higher degrees of specialization. This transformation may leave little of the essence from which the techniques came. At many of the stages along this evolutionary path, malleability may be the factor that determines whether a particular instantiation will be a candidate for further duplication or be discarded in favor of a younger sibling.

6•7

Like people, analysis techniques come into the world, grow up, get old, and sooner or later die.

A class of techniques important to seismologists is derived from the realm of *signal processing*, and is roughly categorized as *time-series analysis*. This extended family of processing schemes can be sub-categorized in various ways, but the building blocks include things like dot products, vector products, FFTs, weighted sums, and matrix multiplies. These are the fingers and toes of the individual packages. In the same way that you can tell girls from boys, some people think that you can differentiate between high-volume *production* packages and research tools. Does this distinction mean that the parts are never interchangeable? Given that creative surgery is often the genetic engineering methodology of choice for software developers, we might ask, "Can we graft the toe of the fullback onto the foot of the ballerina and expect good results? Will we get a female track star or end up with a clumsy dancer?"

PARTS IS PARTS

A key method for increasing the predictability of surgery is to make processing modules work more like spare parts than fingers and toes. Over the years, computing has generated many helpful models for the fabrication and interconnection of these parts—among these are processes, procedures, libraries, **include** files, and macros. Tools for creation, routine maintenance, repair, and replacement also abound in language compilers, linkers, editors, library managers, and file systems. It should be clear, then, that among the important features of an operating system used for development are the availability, generality, flexibility, and portability of these models and tools.

An ongoing project comes to mind that illustrates how some of the features offered under UNIX are helpful when applied to a specific class of scientific data analysis. In the August, 1985, issue of UNIX REVIEW, Paula Hawthorn describes array processors as *backend machines* that are used to offload specialized tasks from the general-purpose hosts that they serve. An increasingly popular configuration for these allows you to embed a few small modules in your system by plugging them right onto the host's bus. While less capable than their

larger, standalone counterparts, they are often easier to use and less expensive to purchase and maintain. We recently purchased one of these and have begun to put together some of the building blocks to do time-series analysis with it.

After installing the hardware, we began to unpack the software that came with the unit. Since several people were to work on the project, we had to try to find a sensible home for all of the various pieces, but we discovered that some of the objects we had to deal with were a little foreign to our experience. The compiler and linker supplied for the development of AP microcode followed UNIX conventions closely enough that they could be installed in an *official* place for all to use. But what were we to do with all those things called *task files* that the compiler and linker produced? And what about the mini-operating system for the AP that has to be downloaded with each task in order to run?

During the previous several years of working with UNIX, we had adopted a uniform directory template for software development projects. Once we understood where task files fit and knew what other pieces of software would need to find them, we easily determined how to extend this system to accommodate them and even how to build *makefiles* to provide for their maintenance. Because the mechanism embodied in the **make** utility is so general, we instantly had a powerful tool at our disposal for dealing with what turned out to be a relatively complex and recalcitrant hardware/software subsystem. This turned out to have hidden benefits that showed up not long afterwards.

Following a few weeks of dealing with the all too common problems of incomplete and misleading documentation and wrestling with the hidden eccentricities that all new systems exhibit, we became fairly confident that we would be able to produce some spectacular processing by the time we received a visit from the people funding the project. Since plugging in the unit worked in much the same way as a floating point accelerator, it probably was not illogical for the clients to assume that the unit would be installed in the system in such a way that other software could continue to function in its absence, with only a speed penalty to pay. While attempting to explain to these clients why *the world really didn't work that way*, it occurred to me that if we took a larger view of our objectives, maybe the world really should (and could) work that way.

8•9

UNIX is constantly evolving and most of us in the scientific research community hope it will never lose its ability to do so.

One of my *golden rules* is that unless a really significant improvement in functionality or capacity can be achieved by *hacking the kernel*, we don't. If you consider device drivers as part of the kernel, however, I violate my rule routinely. For this reason, we rejected the idea of developing a full implementation using the trap mechanism or of modifying the C system libraries. We did so because we felt these approaches would be too intrusive for our tastes. Instead, we investigated the method of substituting a *funny* device driver for one we used at the time to talk to the AP. Although we knew this probably would make the kernel grow substantially larger, it looked like an attractive idea.

What we finally settled on, though—for the time being at least—was a parallel library approach. In order for a processing module using the AP to interface with the rest of the system, it must be wrapped in some C *clothing* (which could be woven together using Fortran, or any other language for that matter; the point is that one must have software that runs in the host). By devising a naming convention for accessing task files, we were able to implement modules that could be emulated exactly by host-only software that had been carefully placed in a parallel library, with entry points accessible by identical calling sequences. Had I the source to the AP linker and the time to do it, I truly would have liked to make the linker understand the files produced by the **ar** utility, thus making the job of building task files measurably easier.

The organization of some of the processing schemes we have explored suggests that, for some applications, a host module may want to download several task files successively. For this reason, we might have explored archiving task files into libraries that could be accessed at runtime. Alternatively, since AP code can be expressed in a relatively compact fashion, we also could have devised a scheme under which it could be viewed by the host software in much the same way that compile-time initialized data is viewed (this is the sort of interface provided with many APs). If we had adopted symbol

tables, procedure calls, and file formats that were consistent with those used by existing tools, we might have been able to *trick* these into helping us with the scheduling and management of AP processing functions.

CONCLUSIONS

Naturally, the examples offered here fall short of a thorough analysis of the applicability of UNIX to scientific applications—even within the restricted set of problems they address. UNIX is constantly evolving and most of us in the scientific research community hope it will never lose its ability to do so. Glittering generalities about the system's usefulness when applied to a large class of problems (such as *real-time* applications) are almost certainly ill-advised, but an understanding of the specifics of the application at hand may well lead to a UNIX solution.

What can be said of the solutions we eventually settled on? Were we able to achieve acceptable data acquisition throughput at an acceptable cost in hardware by using UNIX? Yes, given the other goals of our project. Were we able to sustain a high-volume processing environment under UNIX? Yes, given the flexibility required of our approach. Can specialized versions of UNIX that have been tailored for specific environments still be considered UNIX? This can be a religious question, since what you like about UNIX will largely determine what you think UNIX is or should be. Is UNIX really *grown up* enough to support large-scale scientific research, given the variety of processing constraints scientific problems impose? I *think* it is, but only time will tell.

Mr. Goff is Vice President for Computer Applications at Science Horizons Inc., a California research firm. Before helping to found this organization, he served as Staff Scientist and Senior Systems Engineer for Systems, Science and Software (now S-Cubed) for approximately 11 years. During this time he was a key contributor to software systems now in use at the Center For Seismic Studies, a UNIX-based, DARPA-sponsored seismic research facility. ■

UNIVERSITY OF CALIFORNIA, BERKELEY

COMPUTER FACILITY MANAGER

World renowned University is recruiting for an experienced individual to serve as Administrative/Technical Manager of a large academic computing installation with a multi-million dollar budget and a staff of 40 programmers and technicians. Coordinate departmental staff and faculty on campus-wide networking and assist faculty researchers with the planning of new equipment.

REQUIRES:

Proven experience in managing comparable operations and technical staff, preferably in an academic or institutional setting. Substantial budget/fiscal experience to develop and monitor multi-million dollar operations is essential. Must possess knowledge of technical requirements for computer installations. Candidate will possess a UNIX background and knowledge of other major operating systems.

SALARY: $47.7K—$57.6K and excellent benefits program.

UC Berkeley
Personnel Office
2539 Channing Way
Job #06-148-11
Berkeley, CA 94720

(415) 642-2348

UC BERKELEY IS AN AA/EEO EMPLOYER

Circle No. 291 on Inquiry Card

BLAST®

Communications Software for
Micros
Minis
Mainframes

WANG
Data General
AT&T
PRIME
NCR
apple
IBM
digital
HEWLETT PACKARD
MS-DOS
CP/M
UNIX

Any computer with BLAST can talk to any other computer with BLAST, the universal file transfer software linking many different computers, operating systems, and networks. No add-on boards; use any asynchronous modems or direct-connect for fast, error-free data transfer, even via noisy phone lines, satellites, LANS, and packet networks.

$250/micros **$495-895**/minis **$2495 up**/mainframes

Communications Research Group ***1-80024BLAST***

8939 Jefferson Hwy Baton Rouge, LA 70809 504-923-0888

Circle No. 290 on Inquiry Card

Richardson, TX, seems like an unlikely locale for a guy from Brooklyn, but Steve Wallach seems to have adjusted to his new home. As the Vice President of Technology for an expanding, young company, he has taken part in the area's growth—and as a fellow with big ideas, he has demonstrated a Texas-style appreciation for the proper scale of things.

It was a little over a year ago that Convex unveiled the C-1, a 64-bit integrated vector processing system said to offer a quarter of the power of a Cray for a tenth of the cost. The numbers the C-1 has put up, in fact, are impressive: able to handle up to 128 MB of memory, the machine has been benchmarked at 60 megaflops.

The notoriety this innovation has brought is familiar stuff to Wallach. Indeed, it was Wallach's own notoriety that helped launch Convex. As one of the featured characters in Tracy Kidder's best-selling The Soul of a New Machine (Little, Brown, 1981), Wallach has long had a reputation good for opening doors.

In the book, Wallach was portrayed in his role as the principal architect behind Data General's 32-bit Eclipse supermini series. Later, he served as Product Marketing Manager for Rolm Corporation's 32-bit MIL-spec minicomputer.

To explore the suitability of UNIX for number crunching, UNIX REVIEW asked Rob Warnock, who is himself an independent computer architect with nearly 20 years of experience, to ask Wallach about some of the problems that already have been dealt with—as well as some of those that have not.

THINKING BIG

An interview with Steve Wallach

REVIEW: *There seem to be a large number of companies producing what you call "affordable supercomputers". Why is that?*

Photos by Don Johnson

WALLACH: The reason is actually fairly simple. There's a phenomenal gap in the market. When we started this company [Convex], the companies making 32-bit superminis were talking about how the next big market was going to be office automation. It was clear that they weren't going to do anything to solve the problems of the people trying to do simulations. The folks in the scientific market just aren't interested in office automation and menu-driven editors. They want their Fortran and C programs to run fast. So we saw an opening.

New companies tend to be most successful when they're able to create a new market. Tandem created a market. Apple created a market. And we're creating a market. Much like those other companies, we've identified a niche and we're right on it. Now—like in everything else—once you see a good thing, everyone immediately follows.

REVIEW: *Why did you feel it was necessary to go after the low end instead of offering full Cray performance at a better price?*

WALLACH: There are 110 Crays and 50,000 VAXen (or VAX-class machines). It's a lot easier for the customer to get capital authorization for $500,000 than for $10,000,000. DEC and Data General have conditioned the market to budget $500,000 every year for a new machine. Cray, though, has not conditioned the market to spend $10,000,000 every year. It should not be forgotten that the first Crays were bought by the DoD and DOE. Even for an oil company, $10,000,000 is a lot to spend.

REVIEW: *Why didn't networks of supermicros or superminis satisfy the market requirement?*

WALLACH: A woman gives birth after nine months. There aren't any shortcuts; nine months is what it takes. At some point, your program performance degrades to the level of your most constrained component. You only can go as fast as the slowest chip.

At some point, your program performance degrades to the level of your most constrained component. You only can go as fast as the slowest chip.

Micro architectures simply are inappropriate for a lot of the applications out in the market. To be quite honest, a lot of the micros are just overgrown 8-bit and 16-bit machines, but people still try to run 64-bit problems on them.

There's also another issue. Everyone talks about CPU performance these days, but the majority of applications out there are memory-bound and I/O-bound. It's important to know how much main memory you have and how fast your I/O is. That may have more of an effect on your performance than anything else.

REVIEW: *Does Amdahl's Rule apply in the computer environment? [Amdahl's Rule holds that for each instruction per second a machine should have one byte of main memory and one byte per second of I/O bandwidth.]*

WALLACH: It applies to every machine, but everybody seems to get hung up on the CPU. Let me give you a simple example: I have one machine that cranks out 20 MIPS and another machine that does 10 MIPS. The 20 MIPS machine has 20 MB of memory, in keeping with Amdahl's Rule. The 10 MIPS machine has 100 MB of memory. My application happens to have 100 MB of data, which is to be referenced several times during the execution of the program. In the latter case, that means that every time I make a reference to memory, my data is always there. On the 20 MIPS machine, 80 percent of my references are going to go to disk, which means 18 msec access time per access. Guess what? The 10 MIPS machine runs the application faster.

Applications and the people who buy computers are get-

ting more sophisticated. Rather than just running CPU benchmarks, now there are system-level benchmarks that look at I/O and determine how sensitive it is to physical memory.

REVIEW: *The second part of Amdahl's Rule holds that you should have one byte per second of I/O for every byte of memory.*

WALLACH: Look, at one point there was a thing called Gross's Law. It claimed that you needed to spend the square root of price to double your performance. Now, of course, we know that's bullshit. Just compare the VAX and the MicroVAX II. Semiconductor technology obviously has trashed that law. With higher-performance I/O and higher-performance main memory, I think the rules have changed. Like with the Cray 2 and its two gigabytes of physical memory: if you put everything in memory, who cares about I/O?

REVIEW: *Are you already feeling a push to put more than 128 MB on your system?*

WALLACH: Yes, a strong push, in fact. When we started the company and said we were intending to build a system with 120 MB of memory, people said, "God, what a waste of time and effort! Who in hell is going to buy that? Just think of the expense!" That's because no one anticipated that the price of 256K RAMs would fall as much as it has. We now hear that, "Customers lust for memory."

I always explain this by telling a joke: A guy walks up to an Indian who is saying, "Chance, chance." So the fellow asks, "'Chance'? I thought Indians always said 'how'?" "No", says the Indian. "Already know how, just want chance." Supercomputer users always knew what they could do with more physical memory if they had it, but they never got the chance because—unless you worked for the government—you couldn't get your hands on a machine with a gigabyte of memory. Now, though, we actually have sold several machines with over 100 MB of memory.

Thus, the question becomes: do you really have to have all that memory? Yes, even for UNIX. This is interesting: 4.2BSD has a notion of disk cache that keeps you from having to go to disk if your block data can be located by the software that maintains the cache. With all this physical memory, we can make the disk cache as big as we like, so whenever we run up against I/O benchmarks, we just define a disk cache large enough to keep us from having to go out to disk. As a result, our machines have screamed through benchmarks. Some people cry, "Foul! That's not a fair benchmark because I can't do that on my VAX"—to which, of course, we respond, "Right". Then we smile and don't say anything more.

REVIEW: *Besides disk cache, what aspects of UNIX have you found well suited to your needs?*

WALLACH: Most things are fine. But the I/O structure is really lacking for a Convex-class machine.

REVIEW: *What aspect of I/O is a problem? The fact that it's synchronous?*

WALLACH: Yes. And to deal with that, we've added disk striping—just like you now have on a Cray. Striping allows you to take one disk file and make it go across multiple spindles. For example, we can get approximately 1 MB a second out of a single Fujitsu Eagle. A file striped across four Fujitsu Eagles, though, can get 4 MB a second of I/O. We've also added asynchronous I/O, meaning that if you do a disk or tape reference, you can keep going and use the signal mechanism of 4.2 to synchronize yourself. That's an issue that always comes up.

REVIEW: *Why did you choose 4.2 over System V?*

WALLACH: When we started the company in September of 1982, we knew we wanted to put together a virtual memory machine. Berkeley had a virtual memory system; System V did not. We wanted networking and TCP/IP: 4.2 had it; System V did not. And, let's face it, 4.2 traditionally has focused more on scientific applications than on commercial ones. Since we're after a scientific customer base, 4.2 made sense. We just made a business decision.

REVIEW: *So is 4.2 that much better suited to scientific applications?*

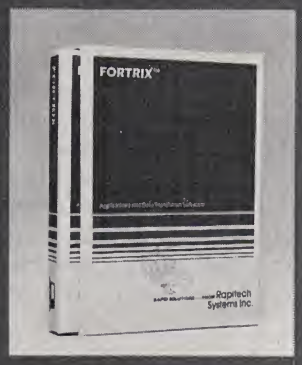
WALLACH: At a base level. But neither 4.2 nor System V has the capabilities that a lot of people want. They don't have the disk striping or asynchronous I/O—and they lack a lot of real-time features like pre-emptive scheduling and the ability to lock pages into physical memory. What it boils down to is that the majority of scientific users are accustomed to their VMS, CDC, or Cray operating systems. Now, when they look at UNIX, they say, "This is great. But I'm used to these five features. Put them on and you've got a sale." They don't care about UNIX, SCHMUNIX. It's features, functions, and benefits that they want.

REVIEW: *I've always thought that I'd be happy if I could get my hands on a UNIX system that had TOPS-10 real-time features.*

Without FORTRIX,™ moving up to 'C' can cost you a bundle!

The bundle we're referring to consists of your existing FORTRAN programs and files. Costly items you'll have to discard when you move up to C, **unless** you save them with FORTRIX™!

Here at last is a program that automatically and rapidly converts FORTRAN code to C code, allowing you to salvage your FORTRAN material at approximately 600 lines per minute. This incredible speed allows a single programmer to convert, debug and put into operation a typical 50,000 line package in only one to two weeks. Plus, the resulting "C" program will run 15% to 30% faster than the original FORTRAN program, while occupying 35% less disk space! And the system even helps you learn coding in C language as you compare your own familiar FORTRAN programs with the corresponding C language programs generated by FORTRIX!™



There's a complete selection of FORTRIX™ versions to suit the full range of user requirements: Original FORTRIX™-C, which translates FORTRAN code to C code, allowing input data files to remain fully compatible with your new C program; FORTRIX™-C+, with the added ability to handle COMMON and EQUIVALENCE statements, character handling and direct I/O; FORTRIX™-C′, the complete FORTRIX™-C+ package configured for non-UNIX* systems including VAX/VMS; and FORTRIX™-C/micro, standard FORTRIX configured for use on the IBM PC and compatibles.

FORTRIX™ has already been installed on 26 different brands of hardware, so whichever FORTRIX™ version meets your needs, you can be sure it will exceed your expectations in terms of speed and cost savings realized. Why not act now to save **your** bundle? Get full technical details, plus references from among over 100 satisfied licensees, from Jim Flynn at (212) 687-6255, Extension 44, or write to him at Rapitech Systems Inc., Dept. A2, 565 Fifth Avenue, New York, NY 10017.

FORTRIX,™ Fortran-to-C Conversionware℠ from

Rapitech Systems Inc.™

RAPITECH

Telephone (212) 687-6255/Telex 509210

NEW RELEASE! NOW, FORTRIX FOR VAX/VMS AND MS-DOS!

*UNIX is a trademark of AT&T Bell Laboratories

Circle No. 256 on Inquiry Card

WALLACH: Then you might be interested in knowing that we have a customer who's bringing up a TOPS-20 shell on top of UNIX. There are a lot of programmers who are used to TOPS-20.

REVIEW: *In light of that, why has UNIX taken over the supercomputer so quickly?*

WALLACH: It's very simple—standards. In any DP shop, the biggest life-cycle cost is Joe in the software department. When people come out of school today, they tend to know UNIX. If you're trying to hire programmers, you pay attention to that because you know if you have UNIX, your new hires are productive after a week. If you have some proprietary operating system, you're looking at a six-month training cycle. That and the transportability of code are the name of the game for UNIX.

REVIEW: *At one time, many vendors were afraid of standard operating systems, fearing that their customers might leave. Has that changed?*

WALLACH: The bigger you are, the less you like standards because you want to lock people in. Standards mean that the lock these companies once had isn't a lock any more.

REVIEW: *Has your commitment to UNIX caused you any business problems?*

WALLACH: The only problems relate back to the fact that while these people want to use UNIX, they also want maybe 10 functions from their old operating system—like one for tape handling, for example. You know, UNIX is not very big on handling magnetic tapes. But certain industries are very dependent on tape—the geophysical [oil] industry, for example. UNIX also does not support IBM communications, but, believe it or not, there are still lots of people out there who want IBM communications. So the problem is that while UNIX is a very good development system, it has some real drawbacks in a production environment. Most of the time we've spent on UNIX has been used to build up production and real-time capabilities—as well as some system management features so that customers don't have to hire a Ph.D. from UC Berkeley to handle their system administration.

One of the things that we've found as we've added these features, though, is that people will say, "That's not within the UNIX philosophy. That's not UNIX-like." But our idea is that if there's someone out there who has hard cash and wants to buy some machines that have certain features, we're going to say, "Yes, sir."

Everyone talks about CPU performance these days, but the majority of applications out there are memory-bound and I/O-bound.

REVIEW: *What about compatibility?*

WALLACH: These extra features always are extensions, not modifications.

REVIEW: *Besides the need for extra features, have you come across aspects of UNIX or C that have caused you problems? I would imagine, for instance, that you're more used to working with Fortran arrays than with C's pointer types.*

WALLACH: That's correct. We are now doing a vectorizing C compiler that will be available shortly. It's an adaptation of our Fortran, so we've developed a new compiler technology for it.

REVIEW: *So you are actually looking at how people use C and then optimizing that?*

WALLACH: Yes, that's very important, in fact. A lot of our optimizations and a lot of our features come from application software. Rather than figuring out what to do next, we let benchmarks and user code drive the functionality. We're a very market-driven company. I can point to features in the architecture, the compiler, and the operating system, and then point to pieces of major third-party software that stress these features. Asynchronous I/O is in all the finite element codes, like NASTRAN and ANSYS. Striping is useful in fluid dynamics code, as seismic interpretation is in reservoir models.

It's my opinion that the companies that succeed will be the ones that recognize what their customers need. Nobody wants or can afford to hire any more programmers. Companies want vendors to produce tools that will allow them to increase the productivity of the programmers they already have.

I have an interesting story about that. Among many other optimizations, our compiler does what is called "dead code elimination". That is, if a piece of code is never executed, we can detect it. Every so often, someone will bring in a benchmark that was written 10 years ago, and—like everything else—has been patched for 10 years. We'll run it through our compiler and get back the message, "Line so and so, dead code removed". The fellow will look at that and ask, "What does that mean?" When we tell him that the code was never executed, he'll go back and start tracing and sooner or later he'll say, "I'll be darned, you're right. I've been maintaining a piece of code for 10 years that's never been executed."

The VAX is a very slow "eye opener". It was built as a minicomputer, you know.

When we first started, the only thing we pitched was MIPS and megaflops. Don't get me wrong—we still do that. But, more and more, the buy decisions are being made on the basis of productivity issues.

Not to downplay the importance of hardware, but we now have more software people than hardware people. That's basically where you have to focus. Hardware people, of course, can be much more productive now because of CAD. In fact, compared to the [Data General] MV-8000, we had less people working on the design of this machine [at Convex]—even though it's probably an order of magnitude more complex than the MV-8000. That's a very good milestone in my book. I should point out that in all my years of computer development, I have never worked with a more talented or gifted team of people than here at Convex. In 15 months, the designers had a prototype working with full VLSI—running UNIX and executing code generated by our Fortran compiler.

REVIEW: *Are there any aspects of UNIX that seem to cause problems for large-machine architectures? Right off hand, I would think that heavy use of character-at-a-time I/O would cause a lot of context switching.*

WALLACH: We put a sledgehammer to that. All the character I/O is off-board on IOPs [I/O processors]. One of the best experiences we had with that occurred on a prototype. We were printing out something when the CPU stopped—but we didn't know it. We were still printing voluminous lines and pages. That's because there are no device drivers in the CPU—they're all on IOPs. So what happens is that when you print on a line printer, the kernel executing on the CPU supplies a byte number and byte count. It then interrupts the IOP, and leaves it to do its own work. That's what an MC68000 is really good for, as opposed to a high-speed processor. We can use 68000s to totally offload all disk I/O, all tape I/O, and all character I/O.

REVIEW: *Does this make it possible for customers, for example, to write their own device drivers without having to learn your machine language?*

WALLACH: Yeah—and what is more, all the device drivers are written in C. Even the diagnostics are written in C. You can go from disk to tape over the I/O bus without using the CPU. The thing is: when we built this machine, we built a system. My experience has taught me that, while everyone focuses on CPUs, they let the I/O go by the wayside. But you've go to hit it with a sledgehammer—which by the way is something DEC hasn't done yet. The VAX is a very slow "eye opener". It was built as a minicomputer, you know.

REVIEW: *Speaking of minicomputers, has your Data General experience—your notoriety with the MV-8000—been an asset or a liability?*

WALLACH: Actually, it's been a massive asset. Since my life is a living resume, there's very little I could hide even if I wanted to. Companies that we deal with can figure out that we didn't just decide to build this machine yesterday even though they don't actually know what we were doing. There's a big advantage in having a very public resume. You know, the old joke is that when Bobby Thompson hit his home run in Ebbetts Field, there were 30,000 people there, but 10 years later, if you had walked around Brooklyn, you'd have sworn that 300,000 people had been at the game. We've all seen resumes of people we've worked with five years earlier and seen things we know aren't true. It's great to have some credibility.

REVIEW: *The process is called "due diligence", I believe.*

WALLACH: That's right. When we were raising money, the venture capitalists made a run on bookstores to get copies of *The Soul of a New Machine*, because I would tell them, "Look, just read the book. It's fairly accurate."

REVIEW: *You seem to enjoy attacks on the big powers. Does that belong in your resume?*

WALLACH: Absolutely. In fact, the biggest thrill for me is the challenge. I've never backed away from one yet. I'm one of those people who shouldn't be kept around if I'm not motivated. You'd be better off getting a clerk to do the job. I think a lot of people around me feel the same way. We're doing battle now with the big powers, which for reasons known only to themselves have lost their focus on the scientific market. If you're working on something you believe in, are having a bit of fun, and are making some money to boot, who can ask for more?

REVIEW: *Do you find that certain general design decisions tend to live a long time and that you end up applying them over and over?*

WALLACH: In a way, yes. I once met Gene Amdahl at a conference. This was back when he had just announced his 470 at Amdahl Corporation. I asked him if the machine offered IBM 1401 emulation because a lot of the 360s had it. And he responded, "The son should not pay for the sins of the grandfathers." You know, at some point we've got to stop propagating mistakes.

REVIEW: *You said earlier that you had found yourself using some of the features of an APL machine you designed in 1971 in the Convex computer.*

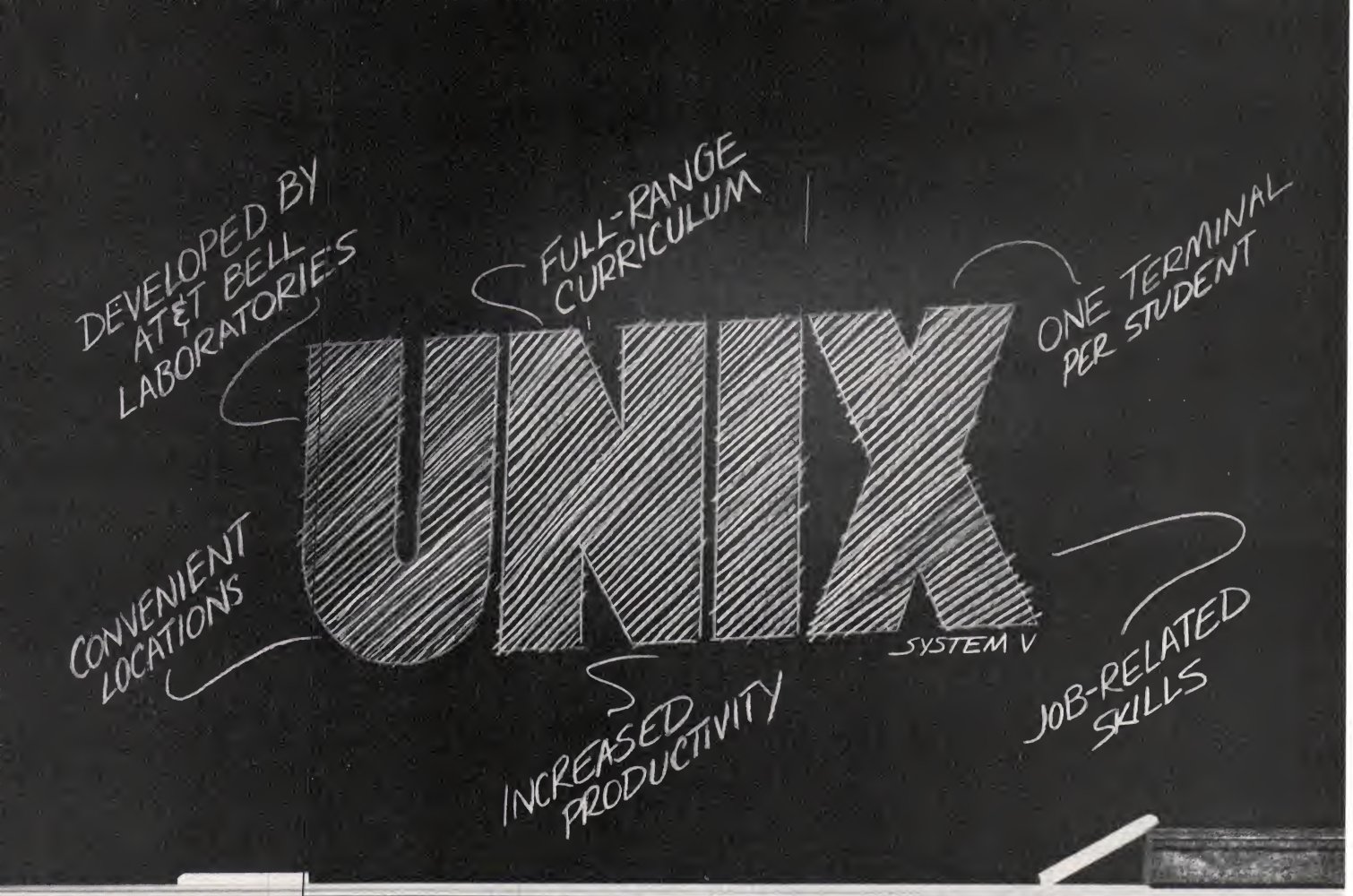
WALLACH: That's right. We used some features—some concepts—because they worked before. You know, if a wheel is round, it can be used by a car. Let's have a round wheel.

REVIEW: *Is UNIX a wheel?*

WALLACH: That's a good question. I think it's more a level and fulcrum than a wheel. The best thing that can happen to UNIX—strictly from a business viewpoint—is for the schism between System V and 4.2BSD to disappear. As a manufacturer, I'd love to see it. It would be beautiful if UNIX were brought under some sort of ANSI control. Then, at least, there would be a defined document not under the control of a single manufacturer. [The IEEE P1003 Committee *is*, in fact, at work on a UNIX standard definition as of this writing.]

REVIEW: *Is there any precedent for such independent control of operating system standards?*

WALLACH: To my knowledge, no. But that doesn't mean it can't be done. Lack of precedents certainly never stopped UNIX. ■



HANDS-ON TRAINING THAT ISN'T SECONDHAND

When you learn the UNIX™ System directly from AT&T, you learn it from the people who develop it. So all the information you get is firsthand.

For over fifteen years, we've been teaching our people to use the UNIX System—which makes us the best trained to help you learn.

The best training starts at your own terminal. That's why, at AT&T each student gets the use of an individual terminal for real hands-on training.

Take your pick of courses from our extensive curriculum. Whatever your level of expertise, from first-time user to system developer, we have a course that will suit your individual needs. And all our courses are designed to teach you the specific skills that will soon have you using the UNIX System to organize and expand your computing system for maximum efficiency.

You also get experienced instructors, evening access to training facilities, and your choice of training centers. We can even bring our courses to your company and hold the training at your convenience.

And because we are continually expanding our courses to incorporate the developments of UNIX System V, you're assured of always getting the most up-to-date information.

So take your training from AT&T. And discover the power of UNIX System V—right from the source. **Call us today to reserve your seat or for a free catalog.**
1-800-247-1212, Ext. 387.

Yes, I'd like some firsthand information on all UNIX System training courses.

Name _____

Title _____

Company _____

Address _____

City _____ State _____ Zip _____

Call 1-800-247-1212, Ext. 387
or send coupon to:
AT&T Information Systems
P.O. Box 45038, Jacksonville, FL 32232-9974

AT&T
The right choice.

©1985 AT&T Information Systems.

THE FINAL FRONTIER

Continued from Page 26

the discussion below.)

- Several implementations of subroutine libraries compliant with Graphics Kernel System are available for use on UNIX systems.
- Many standard UNIX utilities provide for interactive data analysis and display (**awk**, **plot**, **hist**, and **S**, among others).

In addition to these facilities, several other UNIX system utilities can provide significant support during the software development cycle. These include **vi**, **make**, **SCCS/RCS**, and the system's various document preparation tools.

Note that UNIX, as it is commonly delivered, is not able to provide the facilities necessary to support event-driven applications. This does not imply that the system can never be used in such a capacity, but it does indicate that kernel modifications typically are required to integrate a UNIX-based computer into real-time applications.

THE FORTRAN PROBLEM

By now, it should be apparent that the single most important demand made by scientific applications of an operating system is for an optimizing Fortran compiler with a robust, standard Fortran runtime library. The f77 compiler does not satisfy this requirement very well. It is important to look into the reasons for the deficiency, and to see how the problem can be rectified.

UNIX achieved its first notoriety as a system programming and document preparation engine, neither of which require much in the way of floating point support. C, meanwhile, is an excellent systems implementation language, and is the source language of most of the UNIX system. As a result, most work in compiler optimization for UNIX systems is devoted to C.

Development of compilers for different high-level languages on the same system can follow two general approaches: 1) each compiles the source code directly into object code, or 2) each compiles the source code to an intermediate representation that a single-code generator then can use to produce object code. In the first case, the compiler writer can generate object code that takes maximum advantage of the instruction set of the machine; in the latter, the compiler writer can generate intermediate code that maximizes use of the intermediate machine architecture. Despite the portability and economies of scale represented by the second scenario, efficient object code generation is dependent on the richness of the intermediate machine architecture.

UNIX provides many of the facilities necessary to support the computationally intensive class of scientific applications.

The f77 compiler under UNIX uses the intermediate approach for object code generation, taking advantage of the code generator offered by the system's C compiler. Unfortunately, many of the classic programming idioms employed by Fortran programmers are not typical of the way C programs use machine resources. As a result, there is a poor match between the idioms and C's intermediate machine architecture, leading to non-optimal object code for many of the most heavily used Fortran constructs.

One possible solution to this quandary is to convince scientists to use a different language. Much has been learned concerning the use of program and data structures since the first Fortran compilers appeared. The lack of general data structures and a pointer data type often cause algorithms that are really quite simple (when expressed in a modern structured language) to take on the appearance of spaghetti when expressed in Fortran. A new language will not succeed, though, unless it can be shown unequivocally to outperform Fortran in candidate scientific applications. Only then will scientists be induced to accept the startup costs of learning a new language.

Another possible way to increase the appeal of UNIX for the scientific community is to abandon f77 and develop an optimizing Fortran compiler that compiles source code directly into object code. The portability of the UNIX system permits vendors to quickly provide a proven, sophisticated, multi-programming operating system. With an optimizing Fortran compiler, these same vendors would be able to increase their penetration of the scientific and engineering market sectors.

THE FUTURE IS NOW

The major roadblock to a more general acceptance of UNIX in the scientific community is the availability of an optimizing Fortran compiler for each particular hardware architecture. Despite the revulsion purists experience when contemplating such a project, several manufacturers are starting to pursue this approach. This is especially true among the

supercomputer vendors. Note that though the Cray 2 provides a UNIX environment, it makes use of its own optimizing Fortran compiler. The same applies to Convex Computer Corp. and other "affordable supercomputer" manufacturers.

Most UNIX systems provide little of the necessary support for event-driven applications. Some companies have attempted to provide such facilities, but only at the expense of making major changes in the underlying UNIX kernel. It is also true (at least in the experimental physics community) that much of the data acquisition and experimental control performed by such systems is handled by dedicated microprocessor systems running standalone operating system kernels. The communication between these micros and other timesharing hosts typically occurs by way of standard local area networks. Thus, the need for a standard operating system to support event-driven applications is substantially reduced.

Of course, one can always hope that a structured successor to Fortran will eventually emerge. Despite the improved programming environment such a language would undoubtedly provide, its performance will have to be vastly superior to today's Fortran if it is to win general acceptance in the scientific community. In the meantime, we will find that Fortran continues to be heavily used in scientific applications, and that scientists continue to pass up UNIX systems unless they can be shown that UNIX satisfies their Fortran needs in a realistic manner.

Joe Sventek is a member of the Computer Science research staff at Lawrence Berkeley Laboratory and a member of the Computer Science faculty at the University of California at Berkeley. In a previous life, he authored programs representative of both general categories of scientific applications—none of which were crafted or run on UNIX systems. ■

Copyright 1985 by Joseph S. Sventek.

Q-CALC

A superior spreadsheet on UNIX*

As powerful as Lotus 1-2-3*

- large spreadsheet
- many business functions
- complete GRAPHICS package
- translates 1-2-3 models into Q-CALC
- already ported to: VAX, Callan, Fortune, 3B2, Cyb, Plexus, Codata, Cadmus, Masscomp, Sun, etc.
- Ideal for VARs/ISVs

Available since Jan. '84
For more information write/call
Quality software Products
348 S. Clark Drive
Beverly Hills, CA 90211
213-659-1560

*Lotus 1-2-3 is a trademark of Lotus Development Corp. UNIX is a trademark of AT&T.

Circle No. 294 on Inquiry Card

68000 68010 68020 SOFTWARE TOOLS

WE ARE PROUD TO ANNOUNCE THE BIRTH OF THE NEWEST MEMBERS OF OUR 68000 FAMILY . . . YOUR 68020 TOOLS ARE HERE!

TOOL KIT

- 68000/10/20 Assembler Package:
  - Macro Cross/Native Assembler
  - Linker and Librarian
  - Cross Reference Facility
  - Symbol Formatter Utility
  - Object Module Translator
- Green Hills C 68000/10/20 Optimizing Compilers
- Symbolic Debuggers

AVAILABILITY

VAX, microVAX, 8600, Sun, Pyramid, Masscomp, IBM/PC, OASYS Attached Processors for VAX and PC, others. Runs under VMS, Bsd 4.2, System V, MS/DOS, dozens more.

You name it . . .

We provide a "One-Stop Shopping service for more than 100 products running on, and/or targeting to, the most popular 32-, 16- and 8-bit micros and operating systems.

FEATURES

- Written in C; fast, accurate, portable.
- Supports 68000 and 68010.
- 5,000 line test suite included.
- EXORmacs compatible.
- Produces full listings and maps.
- Outputs S-records and Tek-Hex formats.
- Runs native or cross. Extensive libraries.
- Supports OASYS compilers.
- Generates PROMable output and PIC.
- Full Floating Point support.

Over 100 Other OASYS software tools to choose from.

A DIVISION OF XEL Oasys

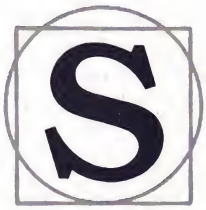
60 Aberdeen Avenue, Cambridge, MA 02138 **(617) 491-4180**

Circle No. 295 on Inquiry Card

DATA ANALYSIS THROUGH INTERACTION

Use of the S system to emphasize human effectiveness

by Richard A. Becker and John M. Chambers



is a language and a system for the interactive analysis of data. The system has applications in any field where data is involved: financial analysis, business graphics, quality control, engineering, and many more. It runs under the UNIX operating system and is described in detail by a 550-page user's guide, *S: An Interactive Environment for Data Analysis and Graphics*, by Becker and Chambers (Wadsworth, 1984). The system is currently used by businesses, universities, and research laboratories. Although it is hard to be precise, we know that there are hundreds of S sites and thousands of users.

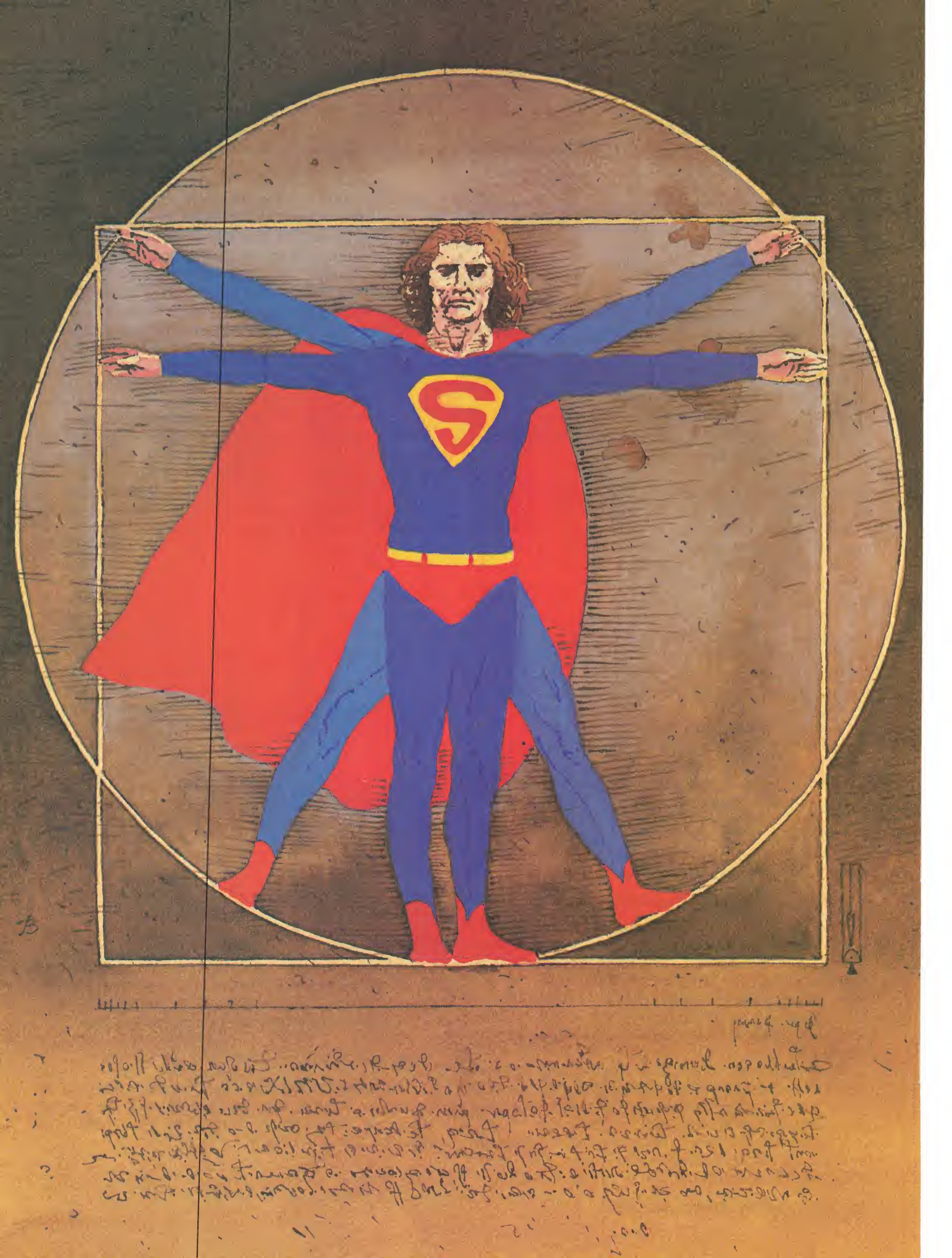
The design goal for S, stated most broadly, is *to enable and encourage good data analysis*. S provides users with an environment that helps them look quickly and conveniently at many displays, summaries, and models for their data. It allows the user to follow the kind of iterative, exploratory path that most often leads to a thorough analysis. By typing simple but general expressions to the system, the user gets immediate, informative feedback, possibly including output on a graphical device. In addition, the system is open to change; even though the S system has many capabilities, a variety of mechanisms are available for extending the system as new applications and techniques appear.

OVERALL ORGANIZATION

An S user types *expressions* that describe the analysis to be done. Some examples can be found in Figure 1. The expressions involve a wide variety of *operators* and *functions* that carry out arithmetic and mathematical operations, statistical analyses, graphics, data manipulation, and other computations. Expressions also use and create *datasets* containing data structures, such as vectors, arrays, time series, and tables. Datasets are automatically accessed by name. The S *executive* interactively parses expressions and controls their evaluation.

The organization of S resembles that of an

Illustration by Robert Williamson



```
  # read a vector of numbers from a file, create data set mydata
mydata <- read("my.data.file")

mydata - mean(mydata)              # subtract the mean from each value

  # Given a matrix of predictor variables longley.x
  # and a response variable longley.y
  # get the residuals from a multiple linear regression model
r <- regress(longley.x,longley.y)$resid

  # compute the residuals
  # larger than the median absolute residual
r [ abs(r) > median(abs(r)) ]
```

Figure 1 — *Some sample S expressions.*

interactive operating system: the executive corresponds to a command interpreter, the datasets relate to files, and the functions can be equated with individual commands. Specific similarity to the UNIX system organization is probably not coincidental, although it was not conscious. There are significant differences, however. The expressions for data analysis need a richer syntax than the commands in an operating system, particularly for algebraic expressions, and data for arguments and results need more structure (commands in the UNIX system operate largely on unstructured streams of bytes).

S was designed in a research environment for statisticians who continually develop new techniques, so it was essential that the system be extensible. Some of this extension (macros and new data structures) can be done within the interpretive S language itself. Other extensions involve the creation of new S functions. Facilities for extension are intended for *users*; they are not restricted to those familiar with the internal workings of S.

EXPRESSIONS: THE LANGUAGE

The user who types expressions to an applications system wants a combination of simplicity and flexibility. Simple requests should be straightforward and brief. At the same time, unusual but sensible requests should not be impossible or unreasonably complicated. Novice and expert users will place different emphasis on the simple or the unusual.

In S, all user commands follow one general syntax: *everything is an expression*. The expressions that are given to S may be as short or as long as is comfortable for the user.

Expressions in S use functional and algebraic syntax, as Figure 1 shows. For users with some background in mathematics, science, or engineering, this syntax is readable and familiar. Extensions to ordinary algebraic notation introduce a few special operators; for example, a colon is a sequence operator such that $x{:}y$ is a vector going in steps of ± 1 from x to y.

When an expression is given to S, it is evaluated. The result may be assigned a name and thus saved as a dataset. If the result of an expression is not assigned or used inside another expression, it is printed for the user.

Algebraic notation (prefix or infix operators, in other words) is natural for functions with one or two arguments. However, data analysis quickly becomes involved with functions that have many arguments. Functions in S can have arbitrarily many arguments that can be specified by either position or name. Typical functions to carry out statistical or graphical analysis will have a few arguments to say what data is to be analyzed or plotted as well as many optional arguments to control details. Options are most easily supplied in the form *name=value*; the options of interest can be specified in any order. Functions return data structures that may have an arbitrary number of named components; thus, functions may have any number of inputs and produce any number of outputs.

One of the most powerful functions in the S language is represented by the subscripting operator. Since S deals with vectors, it is natural that subscripts are also vectors. Thus:

```
x[ 1:5 ]
```

returns the first five values in x. Since it is frequently necessary to exclude observations during data analysis, negative subscripts specify the values to be excluded:

```
x[ -6 ]
```

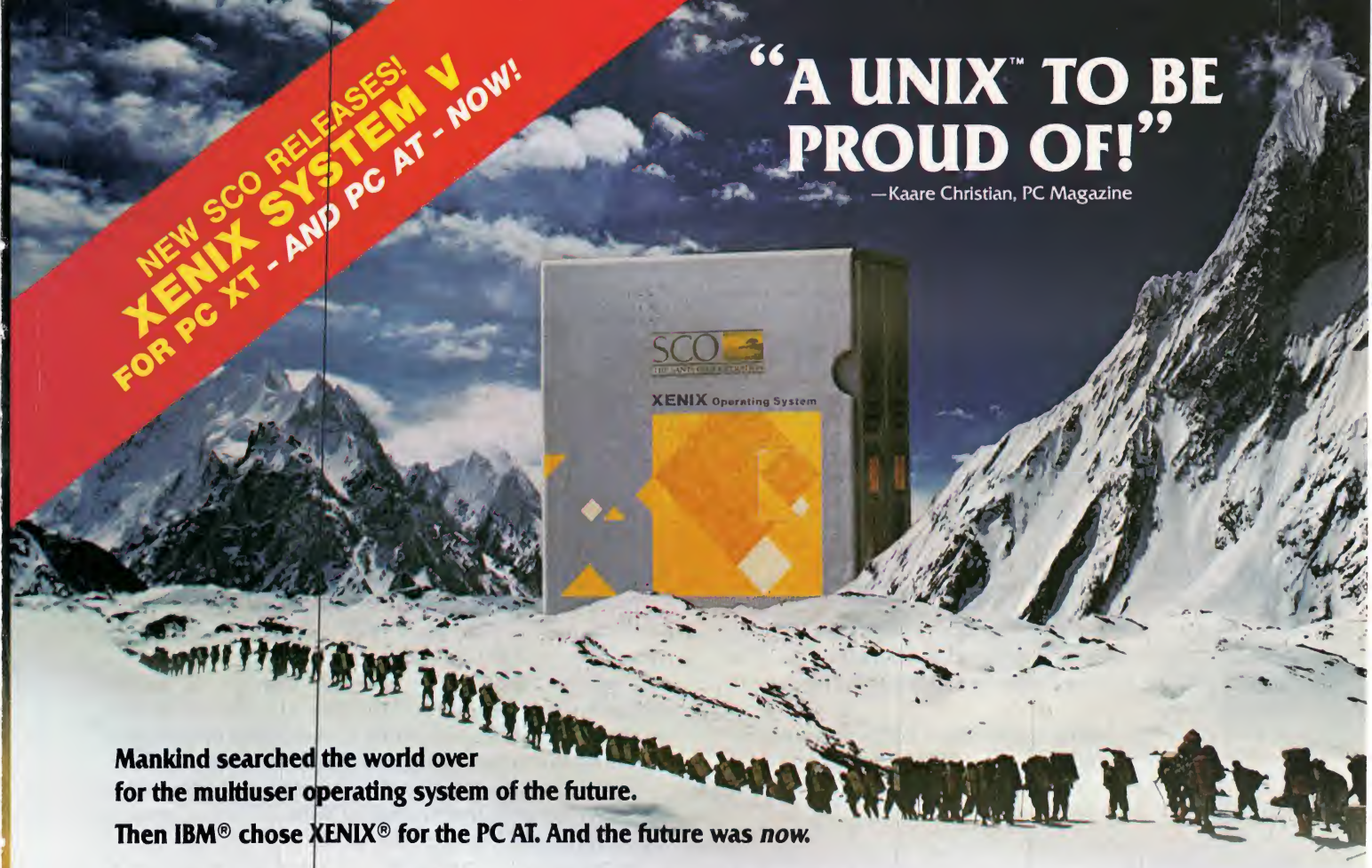
returns x with the sixth value omitted.

Subscripting can also be used to answer database-like queries. Logical expressions used as subscripts cause the selection of data corresponding to TRUE values in the subscript. For example, the query "give the names of people under 25 who make more than $30,000" would be expressed as:

```
name[ age < 25 & salary > 30000 ]
```

The subscripting operation extends naturally to multiway arrays, and in this context an empty subscript denotes all values in that subscript position. For a matrix y:

```
y[ , 6:2 ]
```



Mankind searched the world over for the multiuser operating system of the future.

Then IBM® chose XENIX® for the PC AT. And the future was *now*.

THE SANTA CRUZ OPERATION PRESENTS

XENIX NOW!

AN SCO PRODUCTION IN EXCLUSIVE ASSOCIATION WITH MICROSOFT CORPORATION
THE MULTIUSER, MULTITASKING PC BLOCKBUSTER "XENIX NOW!"
STARRING VISUAL SHELL • MULTISCREEN™ • MICNET • THE BERKELEY ENHANCEMENTS
AND INTRODUCING C-MERGE AS THE MS-DOS DEVELOPMENT ENVIRONMENT
FEATURING WORLD FAMOUS SCO TRAINING AND SUPPORT FOR DEALERS • END USERS • ISVs • OEMs
AND AN INTERNATIONAL CAST OF HUNDREDS OF XENIX APPLICATIONS
INCLUDING LYRIX™ AS THE UNIX/XENIX WORD PROCESSING SYSTEM
PRODUCED AND DIRECTED BY THE SANTA CRUZ OPERATION
SCREENPLAY ADAPTED BY THE SANTA CRUZ OPERATION FROM ORIGINAL STORIES BY MICROSOFT AND AT&T
IN BREATHTAKING SELECTABLE COLOR
NOMINATED FOR ★ BEST DOCUMENTATION! ★ BEST SUPPORT! ★ BEST TRAINING!
★ BEST ELECTRONIC MAIL AND NETWORKING! ★ MOST APPLICATIONS!
★ MOST COMPLETE UNIX SYSTEM!

SCO
THE SANTA CRUZ OPERATION

RELEASED FOR MOST POPULAR PERSONAL COMPUTERS.
APPLICATIONS ALSO AVAILABLE: LYRIX, MULTIPLAN®, INFORMIX®,
LEVEL II COBOL™, 3270 MAINFRAME COMMUNICATIONS.

(408) 425-7222
TWX: 910-598-4510 SCO SACZ

Circle No. 263 on Inquiry Card

M MULTIUSER OPERATION SUGGESTED
XENIX WILL TURN YOUR PC INTO A REAL COMPUTER

©MCMLXXXIV The Santa Cruz Operation, Inc.
The Santa Cruz Operation, Inc., 500 Chestnut Street, P.O. Box 1900, Santa Cruz, CA 95061 (408) 425-7222

UNIX is a trademark of AT&T Bell Laboratories • Lyrix and Multiscreen are trademarks of The Santa Cruz Operation, Inc. • IBM is a registered trademark of International Business Machines Corporation • XENIX and Multiplan are registered trademarks of Microsoft Corporation • Informix is a registered trademark of Relational Database Systems, Inc. • LEVEL II COBOL is a trademark of Micro Focus, Ltd.

Specific similarity to the UNIX system organization is probably not coincidental, although it was not conscious.

returns all rows of columns six through two. As this example illustrates, the subscript operator can also permute data values (here reordering columns six through two).

The function *order* generates subscripts corresponding to a sorted version of its argument. Thus:

```
x[ order(x) ]
```

is equivalent to:

```
sort(x)
```

Using *order* also makes it simple to do passive sorting:

```
name[ order(salary) ]
```

lists names in increasing order of salaries.

The *print* function, implicitly invoked whenever a result is not assigned, represents numerical results to the appropriate number of decimal places and can neatly lay out matrices, time series, multiway tables, and character data.

The function *apply* is able to invoke another function repeatedly on portions of data structures. In its simplest form, *apply* invokes a function on each of the rows or columns of a matrix. Thus:

```
apply( y, 1, "mean" )
```

invokes *mean* once on each row (dimension 1) of the matrix *y* and returns the vector of row means. With other choices for its second argument, *apply* can deal with slices of multiway arrays. Functions can also be applied over hierarchical data structures and ragged arrays.

DATA STRUCTURES AND DATA MANAGEMENT

Datasets in S contain self-describing, hierarchical (list-like) data structures. Datasets are created automatically by assignment expressions; no user control of storage is required. The elementary data structures are *vectors* of numbers, logical values, or character strings:

```
> response
    1.01    .97    3.1    7.21
> response > 2.5
   F   F   T   T
> species.name
 "Setosa"    "Virginica"    "Versicolor"
```

(Here the ">" is the S prompt for an expression).

The numeric *data modes* are "real" and "integer", but for the most part the distinction is unimportant to the user. In S, the value of the expression "3/2" is 1.5, even though in many programming languages integer arithmetic would produce an integer result of 1. A special operator is provided for integer division when it is needed.

There is a special value, *NA* (not available), that can be used to signify missing data. Any arithmetic operations on NAs produce NAs.

General data structures consist of any number of components, each component being either a vector or another general data structure. Each component has a *component name*; syntactically, the component named *Label* of a structure *z* is denoted *zSLabel*.

We designed S so that most users are unaware of the details of data structures, but also so that structures can be defined and manipulated easily to handle new analyses. Simplicity for the user is obtained because all functions that deal with a given type of data structure (for example, matrices, time series, or tree structures from clustering) recognize the structure type by looking for components with certain specific names. Functions that produce such structures as their value simply return structures with the appropriately named components. For example, a multiway array is defined as a structure with two vector components: one named *Data* containing the data values for the array (listed column-by-column), and one named *Dim* containing the extents of the array on each dimension. A 2 by 3 matrix, *x*, with data value $2i+j$ in the $[i,j]$ position corresponds to the following list representation:

```
( "x" STR
  ( "Dim" INT 2 3 )
  ( "Data" REAL 3 5 4 6 5 7 )
)
```

Certain functions make use of a list representation of S data structures to enable structures, or entire

TANDY... Clearly Superior™

The Tandy 6000 lets your office balance the books, track sales and write memos...simultaneously.

For many companies, it's hard to justify the cost of a separate computer for each employee. That's why we designed the efficient Tandy 6000 multi-user computer.

The Tandy 6000 system allows three people to simultaneously access programs and data, and you can expand with up to six users at any time.

With a single Tandy 6000 and printer, you can save time and effort. Your accounting can be processed in one office, word processing in another, and data base management in a third office.

The Tandy 6000 can also help with other departmental functions, like financial planning, inventory, job costing, and sales analysis.

The Tandy 6000 comes with 512K of memory, XENIX 3.0 operating system and a 15-megabyte hard disk drive (26-6022, $5499).

Discover how your business can benefit from a Tandy 6000 multi-user office system. Drop by your local Radio Shack Computer Center for a free demonstration. Ask about our leasing plan, too.

Available at over 1200 Radio Shack Computer Centers and at participating Radio Shack stores and dealers.

Radio Shack®
COMPUTER CENTERS
A DIVISION OF TANDY CORPORATION

Circle No. 253 on Inquiry Card

Prices apply at Radio Shack Computer Centers and participating stores and dealers. Display terminals sold separately. XENIX/TM Microsoft Corp.

New 1986 Computer Catalog. Send me a copy.
Mail To: Radio Shack, Dept. 86-A-789, 300 One Tandy Center, Fort Worth, TX 76102

NAME _____
COMPANY _____
ADDRESS _____
CITY _____ STATE _____ ZIP _____
PHONE _____

The user who types expressions to an applications system wants a combination of simplicity and flexibility.

databases, to be written to files in character form and subsequently read back in.

The ordinary user does not see this structure, however; *x* just appears to be a matrix. When a matrix or array is printed, it is laid out conventionally with no explicit reference to the components of the structure:

```
> x

Array:
2 by 3
      [,1] [,2] [,3]
[1,]     3    4    5
[2,]     5    6    7
```

Matrices and arrays are created and manipulated by a large number of S functions. Data structures such as arrays or time series are so widely recognized that they are considered to be built into the language. Most of the basic functions, such as arithmetic, logic, printing, and plotting, include some special facilities for treating these structures sensibly. For example, the result of adding together two time series is a time series on the intersection of the two time domains.

A broader special class consists of *vector structures* which are data structures that *can* act like vectors but have special structure in addition. Vector structures can be used in arithmetic and, in general, can act as a vector argument to any S function. Arrays and time series are examples of vector structures, but the class is open-ended. Internally, any structure with a vector component named *Data* is considered a vector structure. The *Data* component is the part that acts like a vector when necessary. Functions that operate element-by-element on a vector structure change the data values but leave the other components unaltered. If *x* is the matrix above, *sin(x)* produces a 2 by 3 matrix with data *sin(3)*, and so forth, while $x<4$ is a matrix of logical values.

Functions that rearrange the order of elements, on the other hand, throw away the structure and leave just the data: *sort(x)* sorts the data values in the matrix but its result is a simple vector. Since the original design of S, vector structures have been added to represent such structures as distance measures, categorical variables, and multiway tables. These structures can be used as vectors throughout the language, with no modification of the various S functions involved.

THE EXECUTIVE

The S executive performs tasks roughly comparable to an operating system command interpreter (such as the UNIX system shell). It controls most interactions with the user, parses user expressions, schedules the execution of various functions, and handles interrupts and error recovery.

User expressions are accepted by a parser built using the **yacc** compiler-compiler with a customized lexical analyzer.

The process by which the executive invokes an S function is crucially system dependent. S consists of a large collection of functions (currently around 300). Furthermore, users must be free to write and use their own functions. The facilities of the operating system running S determine how such a collection can be maintained and used in a reasonably efficient way. Operating system constraints have forced us to use several different implementation strategies. For the original version of S, on a Honeywell computer with a relatively primitive operating system (no virtual memory or process control), we wrote our own dynamic loader. Each S function was an overlay, read in by the executive; control was passed by a standardized transfer vector.

When we first moved S to PDP-11s running the UNIX system, the major constraint was the 16-bit program address space. For this environment, we implemented each S function as an independent program. The executive used the **fork** and **exec** operations to start up new processes, and they shared data by means of a common file and noted completion by means of signals.

The current implementation on 32-bit hardware exploits the larger address space to incorporate some or all of the S functions as part of the program containing the executive.

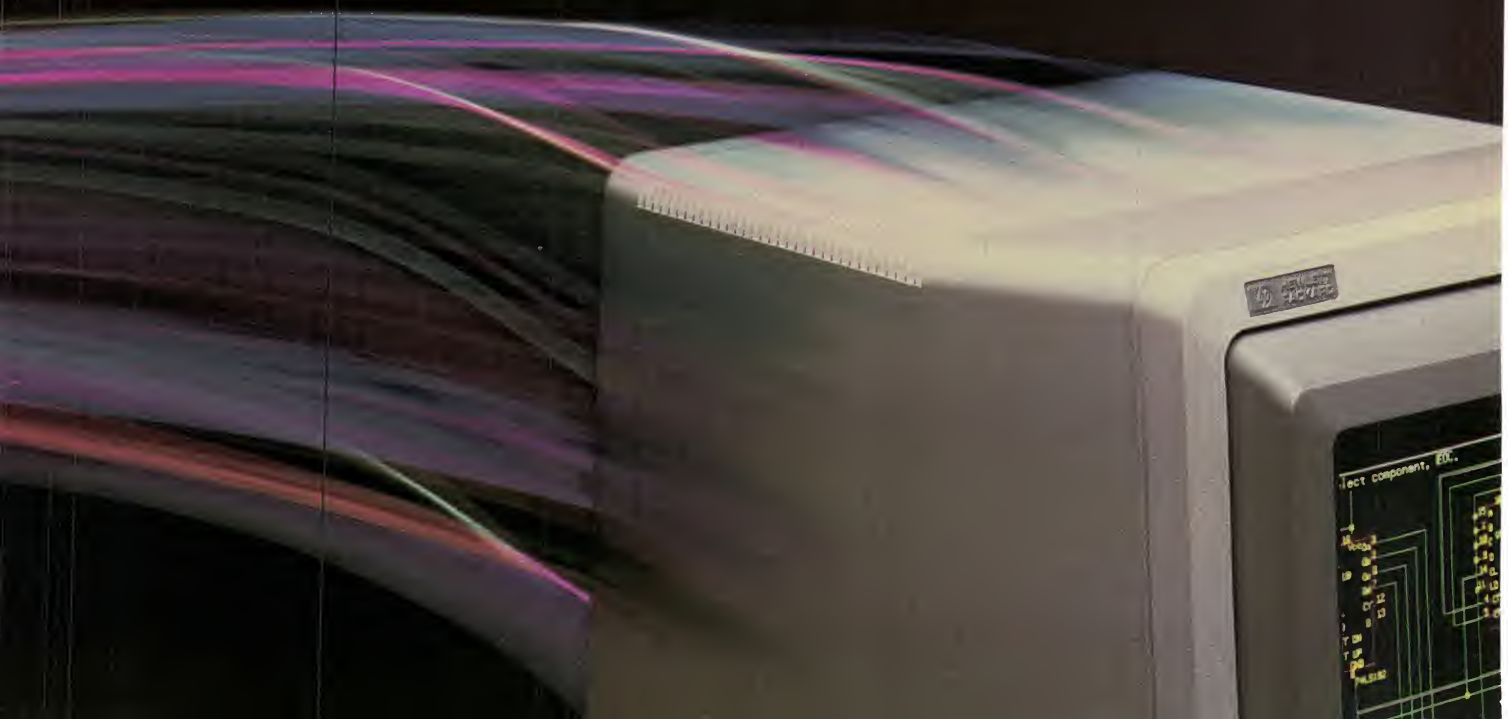
For our goals of flexibility and extensibility, it is essential that these changes in implementation affect only the executive, not the source code for individual functions. Even in the executive, only a relatively small fraction of the code is system-dependent. This code, however, is more crucial to

Name the computer that's so modular and expandable it lets you upgrade from 16-bit to 32-bit processing...

Expand from .5 to 7 Mbytes of memory...

Or go from monochrome display to high-resolution color graphics...

All in a snap...



Introducing the HP 9000 Series 300

The computer that

Starting right now, HP is going to change your thinking on the ways that computers can change. Because now, there's a computer system so easy-to-configure that it meets today's application requirements quickly and cost-effectively, and so modular and expandable that it embraces future application needs as well. Whatever the job at hand — advanced CAD and measurement automation, or word processing, spread sheets, and database management — the new HP 9000 Series 300 is equal to the challenge.

Your pick of processing power.

The Series 300 offers you the appropriate processing power for the job, running your choice of two Motorola microprocessors: the 68010 16/32 bit and the 68020 32 bit. You can start with the 68010 and easily upgrade to the 68020 when more processing power is required. Just as important, you have complete object code compatibility across the product line. So when you change processors, there's no need to recompile.

Changing CPUs in the HP 9000 Series 300 is a snap. You simply plug in a new card set and, with object code compatibility, you shift from a 68010 running at 10 MHz to a 68020 running at 16.6 MHz.

Adding peripherals is easy.

The Series 300 has the built-in interfaces to handle HP's large, fully compatible family of peripherals. There are many compatible monitors of varying resolution, too, so you can go from 12-inch monochromatic display all the way to high-speed, high-resolution color graphics.

UNIX™ operating system is a trademark of AT&T Bell Laboratories.
Ethernet™ is a trademark of Xerox Corporation.

loves changes

In addition, there are a number of HP peripherals for you to choose from: input and mass storage devices, plotters, printers, and more.

Productive programming language options.
You also have a complete set of programming language tools to work with, to help you better meet the needs of your application. For instance, the Series 300 runs HP BASIC, as well as HP-UX — HP's robust version of AT&T's System V UNIX™ operating system. And HP-UX supports industry standard programming languages, too — FORTRAN 77, Pascal, and C.

Link entire systems, not just users.
The Series 300 is designed to be linked with other systems. Your initial application may call for a simple, single-user system. But the Series 300 has what it takes to grow into a sophisticated 100-node LAN based on IEEE 802.3 or Ethernet™. With LAN, the Series 300 can share data with the Series 200 and 500 computers in the HP 9000 family, plus the popular HP 1000 and 3000 family.

Consistent HP quality.
With the HP Series 300, you can count on cost of maintenance below 4 percent, the result of exceptional HP product quality, uniformly maintained with exacting tests in temperature, shock, humidity, altitude, and many others. Couple this with our complete service and support package and you have still more reasons to go with HP.

Call us today!
Choose the system that will change to meet the application requirements of you, your users, and your customers today and tomorrow. Call your local HP sales office listed in the white pages. Or call 1-800-522-FAST (in Colorado, 223-9717 collect) for the number of the sales office nearest you.

Now, get data on-line, 24 hours a day!
For immediate information, use your computer and modem and dial 1-800-367-7646 (1200 baud, 7 bits even parity, 1 stop bit). In Colorado call 1-800-523-1724.

HEWLETT PACKARD

DS15514

Circle No. 248 on Inquiry Card

Users often react to plots by finding the unexpected and using this new information to shape subsequent analysis.

the reliability and efficiency of the system than its size might suggest—adapting the control of such a large-application software system to the features of a non-UNIX system is relatively difficult.

GRAPHICS

Data analysts use plots iteratively as an intimate part of their study of data. The unique role of plots comes from their information content: no other form of output conveys so much information so quickly. Users often react to plots by finding the unexpected and using this new information to shape subsequent analysis. A variety of graphical techniques for data analysis are presented in *Graphical Methods for Data Analysis*, by Chambers, Cleveland, Kleiner, and Tukey (Wadsworth, 1983).

S emphasizes interactive graphics as one of the most important tools in data analysis. Graphics functions in S provide the simple displays that are predominant in statistical graphics—most notably the scatter plot—in a flexible and easy-to-use form. For example:

```
plot(x,y)    #scatter plot
qqnorm(x)    #Normal probability plots
```

The general data structures and expressions in S help to provide graphical output from a variety of sources. Many analyses produce results that define a scatter plot; for example, a *probability plot* shows an ordered set of data plotted against corresponding quantiles of a probability distribution. Deviations from a straight-line pattern help assess distributional assumptions. Rather than duplicating scatter-plot software for each such plot, S functions return as their value a *plotting data structure*, which is passed automatically to the **plot** function to be displayed. The expression:

```
qqnorm(mydata)
```

produces a probability plot of *mydata* against quantiles from the standard normal distribution. Internally, **qqnorm** only generates the plotting data structure and then invokes the scatter-plot function; **qqnorm** needs to know nothing about plotting. The data structure consists of two vector components for the x and y coordinates of the points to plot. Once the probability plot is seen as a data structure, it is straightforward to use this structure for further analysis—by fitting some suitable line to the points in the plot, for example.

The graphical functions are not locked into specific devices because both the user-typed expression and the underlying algorithms are written independently of specific graphic devices. Actual graphical output is produced through a *device driver* that converts the graphics output, at a relatively low level, into commands for a particular device (see Figure 2). The commands are passed from the function to the device driver by means of a set of pipes. Drivers exist for ordinary printing terminals and a range of interactive plotting terminals. A driver is written by implementing routines to carry out a specified set of graphic primitives (such as "draw a line" or "plot a character"), and by providing a definition of the device in terms of basic graphic parameters (for example, the device coordinate system or raster size). Incorporating a new device typically takes a few days or less; the process is straightforward enough that users can write their own device drivers by following the instructions in *Extending the S System*, by Becker and Chambers (Wadsworth, 1985).

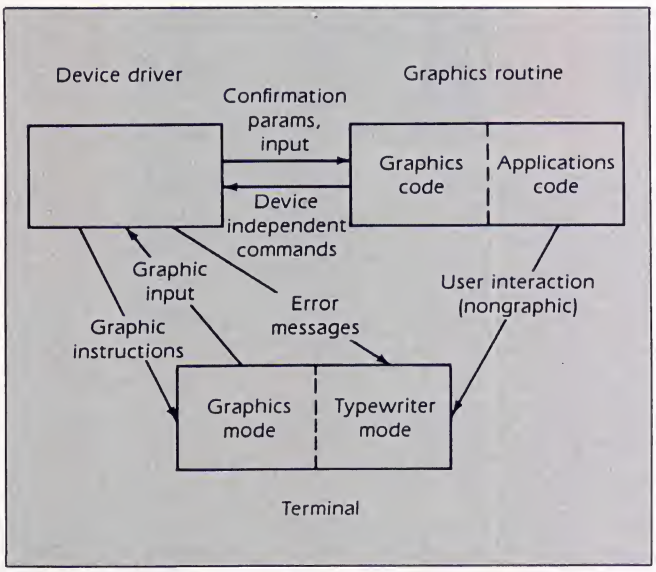


Figure 2 — *Operation of device-independent graphics.*

Only Sperry can make the following four statements.

Our PC runs the XENIX™ system, as well as MS-DOS™.

Our 4 new microcomputers run the UNIX system.

Our new minicomputer runs the UNIX system.

Our Series 1100 mainframes run the UNIX system.

All of which means there is a great deal we can do for you.

For instance, our family of computers based on UNIX systems has incredible transportability for all your software.

And being able to accommodate from two to hundreds of users, it's impossible to outgrow our hardware.

Of course, this linking of all your computer systems can add measurably to your productivity.

And a fast way to find out more is to get a copy of our Sperry Information kit. For yours, or to arrange a demonstration at one of our Productivity Centers, call **1-800-547-8362 (ext. 60).**

*UNIX is a trademark of AT&T Bell Laboratories
XENIX and MS-DOS are trademarks of Microsoft Corporation
©Sperry Corporation 1985.

SPERRY

Circle No. 270 on Inquiry Card

Introducing an idea that makes obsolescence obsolete.

The UNIX* operating system from PC to mainframe.

Resellers: Call Sperry at 1-800-547-8362, ext. 125 to carry the only complete UNIX PC-to-mainframe line.

S was designed using the model of a language operating on complete datasets, interactively, in a nonsequential manner.

TOOLS: THE OPERATING SYSTEM

The complete S system contains about 6000 lines of interface language, 35,000 lines of algorithm language and 9000 lines of C code. Development and maintenance of S by the two of us requires efficient use of time. Our experience is that three aspects of the design particularly affect human efficiency: the *languages* in which programming is done, the *tools* for maintaining the application system, and the *operating system interface*.

We developed our own interface language and algorithm language. This may have accounted for perhaps 10 to 15 percent of our total effort, but this development has been cost-effective. *Interface routines* describe arguments to S functions, check for errors in arguments, allocate space for data structures, call computational routines, and return results. If interface routines were written directly in a general language like Fortran, they would be much more complicated and error-prone, and all but the most sophisticated users would find it impossible to write their own S functions. During compilation, an interface routine typically expands into a much larger Fortran routine (representing an order of magnitude more lines of source code). Much of this expansion reflects inherent clumsiness in using Fortran to express the argument processing, dynamic storage management, and result generation encompassed in an S function. At the same time, the use of Fortran as an intermediate language is important. We could not re-implement all the basic computational algorithms previously written in Fortran.

The use of software tools is essential for creating and maintaining a system such as S. Compiler-compilers, macroprocessors, and more specialized tools ease the burden of system development. During compilation, the interface language goes through our own simple compiler, two passes of the M4 macroprocessor, RATFOR, and Fortran. Obviously, we are not trying to optimize compilation time. This multistep process, however, does enable us to modify individual steps as our needs change.

Other tools are used to provide specific utilities for S developers. The **make** system for maintaining programs is used to generate the S executive and the individual functions.

For tools to be useful in large applications systems, they themselves should be easily adaptable. For example, our use of **make** is highly specialized. The interface routines and the support programs, whether based on RATFOR or C, all take advantage of special S facilities. We therefore replace and extend **make**'s built-in rules for compiling to include these special features. The result is a customized tool, itself built from a number of tools.

The ease with which tools are put together is also a function of the operating system environment. The UNIX environment is convenient for developing a system such as S, both because of specific facilities and because the operating system tries not to be unnecessarily restrictive. Facilities such as pipes and a flexible command interpreter make the creation of customized tools much easier. The *absence* of complex rules about file formats and interprocess protocols, on the other hand, has meant that our implementation has had fewer barriers to scale than it might have otherwise.

The dependence of the current version of S on its operating system environment involves both the internal dependencies and the use of operating system features in the tools. The dependencies on computer *hardware*, such as machine accuracy, are relatively easy to handle. The large majority of S code passes through Fortran during compilation. Non-portable features, such as the choice of special characters and machine precision, are isolated in the macroprocessing phase and kept in a single file.

The use of Fortran as an intermediate language and the parametrization of machine-dependencies make S source code quite portable. On the other hand, implementing and using a system like S benefits from a good general computing environment. The UNIX system has allowed us to combine and modify tools to put together S. In a more restrictive system, we would have been obliged to provide more of the support environment ourselves. Perhaps most importantly, the UNIX system is being used on a variety of new computer systems, and when that is done, S goes along for free. Because of this form of portability, S currently runs on hardware ranging in size from AT&T's UNIX PC to large IBM mainframes.

HISTORY

Work on S began at Bell Laboratories in 1976 and

S represents an approach to computing that emphasizes the effectiveness of the *human* as the most important design criterion.

an initial implementation on a large Honeywell mainframe system was in use late that year. (This was the machine left over after Bell Laboratories dropped out of the Multics project.) Starting in 1978, a version of S was developed for the UNIX system on an Interdata computer just after the UNIX Seventh Edition port was accomplished on that machine. Since 1981, the UNIX-based version of S has been distributed outside Bell Laboratories by AT&T.

When the design of S began, a group of us at Bell Laboratories considered the statistical software that existed at the time in terms of our goal of good data analysis, particularly in an interactive, exploratory environment. We could ascertain three main approaches to doing statistics on a computer: programming in a conventional language, usually Fortran (this had been our own previous approach); mainframe statistical packages such as BMD, SAS, and SPSS; and a few interactive languages, notably APL. We recognized the need for better use of human resources than was possible when it was necessary for individuals to develop their own Fortran applications, but we found problems with the existing alternatives to Fortran.

Statistical packages arose during the 1960s and were closely modeled on the idea of sequentially processing a series of records on punched cards or magnetic tape. This model has had several bad influences. Good data analysis is highly iterative, responding to important facts observed in the analysis itself. Picturing analysis as processing a sequence of records through a limited set of statistical commands discourages this freewheeling interaction with the data. In particular, interactive use of the statistical packages was either not available or consisted largely of the ability to set up the card deck and run it from a terminal. S, on the other hand, was designed using the model of a language operating on complete datasets, interactively, in a nonsequential manner. A number of modern statistical techniques, like robust estimation, cannot easily be expressed in the sequential form, and are therefore hard to incorporate in some of the packages.

Another result of the batch approach was the tendency to "shotgun" output, printing all the summaries likely ever to be relevant from a particular model or process. Instead, S tries to provide a wide variety of displays, particularly graphical, that can be used interactively to see summaries relevant to a particular user. Graphics, like interaction, was not part of the original design of the mainframe packages. Since 1976, many of these packages have added graphical facilities, but the graphics tend to be viewed as "reports" rather than as integral parts of the analysis. For example, most of the graphics add-ons do not include graphic *input*, which in our opinion is essential for identifying important features observed in the plots.

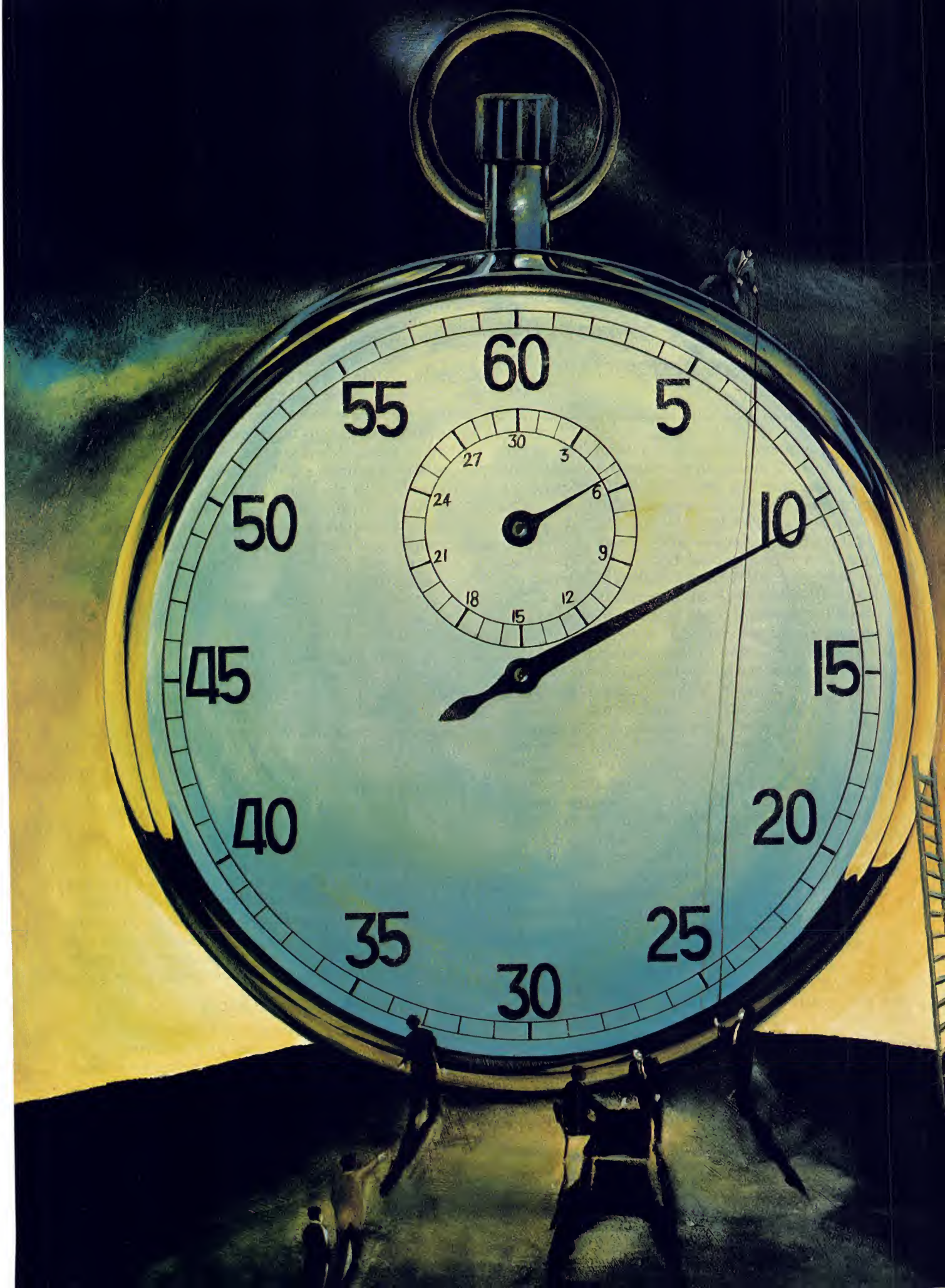
The APL language, while not designed for statistical computing, offered a very different (and, in many ways, more attractive) approach. It was intended for interactive use, with users typing expressions that operated on whole datasets and produced immediate output at the terminal. Users can extend the language by defining interpreted "functions" that can then be used in the same way that primitive APL operators are. These are all features that contribute to APL's usefulness for data analysis, and thus have been incorporated into S. The consistency and functionality of APL's operators are also present in S; in S, however, such operations are normally carried out by functions rather than by operators. The main problems with APL are its syntax, its data structure, and its isolation from other languages.

S represents an approach to computing that emphasizes the effectiveness of the *human* as the most important design criterion, as shown by the emphasis on friendly interactive access to computing, information hiding, and on greater flexibility through delayed binding. Our philosophy is that the effectiveness of the *human* is the most important criterion for the design of any computer system.

EXPERIENCE AND EVOLUTION

A significant contribution to the evolution of S has come from user activities and experience. By far, the majority of our users are not professional statisticians. Instead, they are professionals in other areas who have a need for data analysis, graphics, or other S facilities to enhance their own work. In a number of cases, their specialized use of S has led them to develop, in effect, unique systems for their own specific user communities. This is usually done by creating a set of S macros to

Continued to Page 100


60
5
10
15
20
25
30
35
40
45
50
55
30
3
6
9
12
15
18
21
24
27

UNIX IN REAL TIME

What it takes to make the grade

by Clement T. Cole and John Sundman

As UNIX has moved from the world of computer science research into *The Real World*, its character has been altered. This article explores some of the modifications that have been made to support "real-time" processing and looks at a number of the demands that real-time applications make of the operating system.

Real-time operating systems can be described in terms of seven requirements. It is our contention that UNIX, suitably modified, can fill all seven. Before listing these requirements, however, it's important to understand that they stem from applications—which, unlike UNIX, have no standard definition. (Admittedly, the /usr/group UNIX Interface Standard published in March, 1984, has its detractors, but the IEEE P1003 Portable Operating System Environment Working Group has started work on a better definition.)

Because real-time applications lack a standard definition, this article makes a distinction between a real-time *monitor* and a real-time *operating system*. Even though our distinction is not rigorous, we feel it is important to make if we are to avoid comparing apples with oranges. To this end, we have restricted our comments to applications that require an operating system. We make no claims for the suitability of UNIX as a real-time monitor.

The class of real-time applications that interests us includes the problems that traditionally have been managed by systems like Digital Equipment Corporation's RSX and VMS; Data General's RDOS, MP/AOS, and AOS/RT-32; and Hewlett-Packard's MPE. Our list of seven requirements comes from an analysis of the traits common to these and other similar systems. This seems fair since real-time applications and operating systems have been around for a long time—longer, certainly, than the UNIX system has. Thus, when we

state that a real-time operating system needs multiple processes and quick communications among them, we are simply following a long tradition—not defining the problem in terms of a ready solution. We make this point because UNIX has gotten something of a "bum rap" with regard to its suitability for real-time applications.

For example, in a recent issue of *Computer Design*, three real-time experts—Bill Allen, Collin Hunter, and Bernard Mushinsky—flatly state "Real Time Unix is not a good idea."[13] The accusation is often made that UNIX devotees see UNIX as the solution to every problem, and describe these problems in terms of solutions already built into UNIX. This simply is not the case.

WHAT IS A REAL-TIME SYSTEM?

Let's look at what a real-time system is. There are two parts to consider: a *controlled system* and a *controlling system*. The "controlled system" consists of hardware—such as the sort one might use for a process in a factory or an experiment in a laboratory. The "controlling system" refers to the computing resources—hardware and software—that accept and analyze data from the controlled system. The controlling system sometimes produces data to modify the controlled system.

A real-time operating system, like Digital's VMS or MASSCOMP's Real Time UNIX (RTU) is the nucleus of the software used by a controlling system. As such, the operating system must contain and present to application programs the primitives needed to build real-time applications. An application must be able to *instantaneously* record, analyze, and respond to the raw data and events produced by the controlling system [2]. Note that "instantaneous" is a relative term and that applications described as such must be controlled (more about this later).

> UNIX has gotten something of a "bum rap" with regard to its suitability for real-time applications.

Requirements for a Real-Time Operating System. Real-time operating systems can be described in terms of seven features:

1) Support for the creation, deletion, and scheduling of multiple "processes" or independent tasks, each of which monitor or control some portion of a total application.

2) Communication "channels" between processes to allow small amounts of data to be sent or received, thus allowing two or more related processes to share low-volume information.

3) Data "pooling" or sharing, allowing two or more processes to pass and examine large amounts of data efficiently.

4) Synchronization between processes in an application.

5) The ability to quickly, reliably, and predictably maintain large amounts of data in long-term backing store.

6) Synchronization with external events.

7) "Instantaneous" and predictable response to external events.

Every real-time system embodies these features to a greater or lesser extent. This article attempts to show that standard UNIX, with only minor changes, embodies the first five features on the list. With some other significant changes—which already have been made in specific instances—the remaining two features can also be provided.

A Word on Monitors. A real-time *monitor* is a small, generally PROM-based set of subroutines dedicated to a unique, limited task (an "application" program). *Monitors*, real-time or otherwise, are most often shipped with a "single-board computer" so as to allow an application program to operate the module as soon as power is applied. Often, an applications programmer will not replace a monitor, but instead use it as a starting point, making subroutine or "monitor calls" to it in order to obtain additional functionality. Monitors thus can limit the amount of new code that must be written.

Note, for example, a "smart" Ethernet controller such as the Excelan EXOS-201 or the Communications Machinery ENP-30. Each in itself is a real-time system containing an independent 16-bit processor, memory, and I/O hardware. Both can serve as a controller for the Ethernet and host interface hardware. A small, on-board real-time monitor schedules each portion of the networking task, responds asynchronously to external events (such as the receipt of a packet from the wire), recovers from errors, and performs the normal operations of setting up and breaking down network connections. The controller must perform all of these tasks in real time.

A monitor has no concept of

long-term data retention to a backing store. Neither does it concern itself with providing a user interface by way of a command executive, or with running a number of independent programs invoked by system "commands". A monitor focuses on a single embedded task that can be performed simply and efficiently—like the management of an Ethernet controller.

We place "custom real-time operating systems", such as Hunter and Ready's VRTX or Industrial Programming's MTOS, in the class of PROM-resident monitors since they don't offer all the services of full operating systems. Embedded systems (Ethernet controllers and the like) are often best implemented with dedicated monitors. Such monitors do not have to provide a file system, a command system, or any of the other features one would expect in a full operating system.

The key difference between a full operating system and a monitor is *scale*. As Jim Ready (of Hunter and Ready) has been quick to note, many UNIX vendors embed real-time monitors deep inside their smart controllers [7]. If you define a real-time *application* as one that can respond to raw data and events, you might include monitors. But real-time applications as we define them must accommodate a wide range of activities (often involving several processors), and must be adaptable to large-scale changes, even while analyzing and recording data. This is why we have excluded monitors from our discussion. The services they are designed to provide simply do not compare to the wealth of services supplied by UNIX.

A Data General MV-8000, Hewlett-Packard 3000, MASSCOMP MC-5500, or DEC VAX system could be used for analog data acquisition and encapsulation. An MC-5500 running RTU, for example, can gather up to one million 16-bit samples per second from a controlled system, recording the data in a disk file. As the data is gathered, the system can perform sophisticated analysis. For instance, a user might push data through an array processor or floating point unit with complex arithmetic subroutines such as fast Fourier transforms (FFTs) and linear regressions. The interpreted result then could be displayed in real time on a bit-mapped graphics display screen. Real-time monitors simply are not designed to provide this scale of service.

Originally, UNIX was not written as a "real-time" system but as a "timeshared" system for interactive use.

STANDARD UNIX AND REAL TIME

What is Standard UNIX? For the purposes of this discussion, let us assume that "standard UNIX" means the system call interface as defined by the March, 1984, /usr/group UNIX Standard. We will cite any UNIX features not defined by the standard.

The Real-Time Issue. Originally, UNIX was not written as a "real-time" system [9] but as a "timeshared" system for interactive use. This timesharing heritage is responsible for much of the denigration that real-time UNIX systems suffer today. But if one looks at a "timeshared" system as a "real-time" system, where "instantaneous" response is measured in seconds rather than milliseconds, one might find that UNIX actually has most of the seven features required by real-time data-acquisition applications.

Creating, Deleting, and Scheduling Processes. The support UNIX offers for process creation, deletion, and scheduling is embodied in four simple yet powerful system calls:

- **fork(2)** creates new processes.
- **exec(2)** starts the execution of a different program image.
- **wait(2)** keeps one process on hold until another process is completed.
- **exit(2)** terminates a process.

The UNIX kernel *schedules* which process may execute at any instant; this function is fundamental to system operation. The code that handles it is called the scheduler. Although the standard UNIX scheduler is tuned for timesharing use, it provides one concession for real-time applications—a "priority value" that can be manipulated with the **nice** command.

Of the processes in primary memory, some wait for an event (for example, I/O or the termination of a child process), while others stand ready to continue execution at any time. Associated with each process is a priority that determines the order in which processes will run.

The priority of all operating system processes—including the operating system portion of user processes (system calls)—is greater than that of user jobs. Among these system tasks, disk I/O has a high priority, terminal I/O a low priority, and time-of-day events an even lower priority.

User-process priorities are assigned by an algorithm that weighs the amount of compute time consumed by a process against the amount of real time that's used. A process that has devoured a lot of compute time in the last real-time unit is assigned a low user priority. Because interactive processes are characterized by low ratios of compute-to-real time, interactive response is maintained without special arrangements.

The scheduler simply selects the process with the highest priority, thus picking system processes over user processes. The compute-to-real-time ratio is updated every second. Based on these results, high-priority processes are allowed to preempt low-priority processes—even if the low-priority processes already are running. Further, the scheduling algorithm has a negative feedback character: if a process uses its high priority to monopolize computing resources, its priority will drop. Similarly, if a low-priority process is ignored for a long time, its priority will rise.

Although the priority of a process is calculated by internal operating system algorithms, the user or programmer can make a contribution to this calculation by assigning a *nice* value to a process. This value is set by using the **nice(1)** command, which in turn uses the **nice(2)** system call. The default *nice* value for all processes is 0. Conventionally, this value is allowed to range between 0 and 19. The higher it is, the lower the resulting priority. However, *nice* values alone are not sufficient to guarantee static priority scheduling.

Communications Channels. The *pipe*, a simple, elegant channel for communication between two processes, is one of the most celebrated innovations offered by UNIX. The **pipe(2)** system call offers a method for creating a simple read/write communications channel between two processes. The *fifo* (or named pipe), operates in a similar manner, but does not require that the two communicating processes be part of the same process family tree. These two primitives allow two or more processes to send low-bandwidth information back and forth by *read* and *write* operations on the *pipe*.

Modifications have been made by countless unsung heroes in a myriad of UNIX installations.

Newer implementations of the UNIX system contain communications primitives that supplement pipes. The System V IPC and the 4.2BSD *socket* mechanisms are instances. These are usually used for networking, so they are not discussed at length here. But they are, of course, available for real-time programming.

Data Pooling. Data pooling allows large amounts of data to be shared between or among processes. This can be accomplished by opening a file for access by a number of processes, and also by overlapping (sharing) the data segments (memory address ranges) of different processes. The shared memory technique is adaptable for real-time use.

Shared memory has been offered in a number of schemes. In System V, AT&T provides for it with the **shmop(2)** primitive. This mechanism allows large amounts of data to be accessed by more than one process at a time.

Synchronization Between Processes. The *signal* is the traditional UNIX tool for synchronizing activities in two processes; it allows one process to notify another of an event. The **signal(2)** call provides a software interrupt mechanism analogous to a hardware interrupt. Any process can send one or more signals to any other process so long as the sending process knows the *process id* of the destination process [9]. A process that expects to receive a signal should include a routine (signal handler) that can be called whenever a signal arrives. Certain signals, such as the **alarm(2)** timer signal, can be sent by the system itself.

The **semop(2)** (semaphore operation) primitives of AT&T's releases offer another scheme for synchronizing processes. These primitives provide a mutual exclusion mechanism by using a semaphore operation between multiple processes, and are often used in conjunction with the shared memory (**shmop(2)**) calls. The **semop(2)** calls enable coordination between processes that may attempt to write into the common data pool at the same time.

Reliable Data Transfer to Backing Store. Reliable data transfer to long-term backing store is usually implemented as a file system on disk. The celebrated UNIX hierarchical file system and sophisticated file system utilities represent a higher level of functionality than is strictly required by many real-time applications. A simple real-time system only needs to provide the ability to open, read, and write files—albeit quickly. Real-time applications that perform data logging should spend as little time as possible interacting with the file

system since their primary job is to interact with the system they are controlling. The necessary backing store capabilities are provided by the UNIX calls **open(2), read(2),** and **write(2)**.

NON-STANDARD REAL-TIME ENHANCEMENTS

Having described the features in standard UNIX that are well adapted, or adaptable, for use in real-time applications, we find that only two items are missing from our list of seven real-time requirements— synchronization of processes with external events, and predictable and "instantaneous" response to external events. The rest of this article discusses how UNIX has been enhanced and reworked to provide these features. By "enhanced" we refer to modified features already in UNIX. By "reworked" we refer to entirely new features that have left the UNIX system's original functionality in place.

We hope it is understood that to make UNIX faster (an implied requirement of any real-time system), developers must find and replace inefficient code. Many parts of the UNIX kernel and a fair number of the system's utilities were never designed to be used as they are today; thus their original "tuning" no longer applies. Over time, many of these inefficiencies have been overcome and much code has been retuned. These modifications have been made by countless unsung heroes in a myriad of UNIX installations, but we cannot delve into the details here. Instead, we refer you to the proceedings of the various Usenix conferences of the last 10 years.

Creating, Deleting, and Scheduling Processes. Changes made in this area focus on the scheduler. The scheduler is usually the first thing to be modified when UNIX is tuned for real-time use. That's because most implementations of UNIX use a scheduler that has been tuned to a timesharing load. Timesharing embodies a notion of fairness—each process should be given its proportionate slice of the available computing resources. Real time, though, is autocratic: some processes are more equal than others. The most equal of the lot are not only given the better part of the system's resources, they also must be guaranteed that processing resources will be available to them at any instant. (Other real-time operating systems have employed the concept of *sub-process*, or *task*, to provide a flexibility in scheduling that does not incur the overhead of context switches. However, no UNIX vendor of which we're aware has chosen this approach—although it has appeared in academic incarnations.)

Imagine, for example, an application that monitors the life functions of a cardiac patient. The routine sampling application might take an interrupt every few seconds to monitor changes in the patient's temperature. If a critical event—such as *apnea*—were to occur, though, the application might need to immediately disable temperature monitoring and start monitoring and responding to data returned from cardiac sensors. A second or even a tenth of a second might be too long to wait.

To enhance the UNIX scheduler, the RTU operating system has introduced the concept of real-time priorities. Processes with real-time priorities are selected to run before processes without such priorities. A real-time priority is established by setting a *nice* value in the range -11 to -20.

Real-time processes under this scheme do not have their priorities updated by the scheduler as normal processes do; instead, they are assigned a fixed priority. These processes do not incur scheduling penalties, regardless of whether they monopolize processor time or not. It thus is possible in a timesharing environment to run a process that performs some service at a reliably high priority whenever the need arises. One example of where this might be useful is in the service of a serial line or a graphics processor. If two or more real-time processes are in memory with the same priority, they will run on a *fifo* basis.

Modifications to the scheduler by no means offer the only solution. Charles River Data Systems (CRDS), for instance, has replaced the entire "bottom half" of UNIX with proprietary software better suited to real-time use. The user and programmer operate in a UNIX paradigm, but the internal implementation of the system is "home-grown". The CRDS operating system, called UNOS, looks like UNIX but supports a notion of "process" in both the kernel and user memory space. The internals of UNOS are governed by the use of *eventcounts* [8], and the CRDS system employs a scheduler built around this primitive. The result is a natural cooperation between different processes in which real-time response can be easily attained [3].

Communications Channels. Pipes were originally implemented as a special type of disk file. This made sense on the PDP-11 architecture where UNIX was originally implemented. In that family of machines, memory was limited, making use of the disk for pipes worthwhile but slow. As UNIX migrated to large-memory architectures such as the VAX and the Motorola MC68000 families, use of disk for pipes became less important. Thus, to speed the performance of pipes in real-time

systems, some UNIX variants implement pipes in memory.

Data Pooling. An important problem (from a user program's point of view) with the current UNIX implementation is its use of memory management; programmers may require the ability to manipulate cache (if the system has one) as well as to control those portions of an application locked into system memory.

In most high-performance machines, both an address and a data cache are used to keep the processor running at full speed. When memory is shared, it may be necessary to disable the cache on shared pages. Thus, a user program must have some way of informing the system that sharing is in effect.

MASSCOMP's solution was to supply two system calls, (**pinfo(2)** and **cinfo(2)**), that allow a user to examine main memory use. Other manufacturers have implemented similar calls. Since there is not yet a standard for this type of operation, programmers should be aware that each manufacturer does it differently.

Another problem with memory access is its tendency to make system response less predictable. Virtual memory systems are offered on most modern computers, including the MASSCOMP MC-5000, Apollo DN, and Digital VAX series. Virtual memory is convenient for programming but it wreaks havoc on the "predictability" of a program. In most real-time applications, the programmer wishes to guarantee that at least the key parts of a process have been locked into memory.

Version 7 UNIX contains a facility to allow memory locking, but this feature was dropped until System V brought it back—meaning that a large number of UNIX implementations currently are lacking. Implementations that stress real-time have had to reinstate this facility or move to System V. MASSCOMP's RTU, for instance, provides the **plockin(2)** and **unplock(2)** calls for this purpose.

Another problem with memory access is its tendency to make system response less predictable.

It should be noted, however, that these are "dangerous calls" for a couple of reasons. In the first place, as more memory is locked down, less is left available for general use. What is worse, memory deadlock problems can occur that prevent processes that require memory from proceeding until segments are unlocked. Obviously, a deadlock can destroy the performance of a system or can shut the system down altogether. In a dedicated real-time situation, the programmer should calculate the amount of memory needed and make sure that the machine contains "enough" physical memory to allow the application to be "locked" without causing a deadlock.

Only the system designer can determine what the value of "enough" is for any application. The operating system can enforce quotas but cannot, of itself, guarantee performance.

Synchronization Between Processes. UNIX *signals* are used in a software interrupt scheme originally modeled around the concept of *hardware exceptions*. Processors generate "hardware exceptions" when they receive instructions they cannot perform, such as a division by zero exception. Signals were provided so that programmers could implement exception handlers to allow "orderly termination" if a hardware event occurred. Signals were later extended to allow for a few other "software exceptions", such as an interrupt generated by a terminal interrupt key, like CTRL-C or DELETE.

In standard UNIX, signals are implemented on a process-by-process basis. Each process typically has 16 or more signals that can be handled ("caught") independently. (At least one signal is not capable of being "caught" by the process. This is the termination signal used to abnormally exit and terminate a process with extreme prejudice.) Most recent UNIX systems implement more than 16 signals per process, including at least one reserved for the user.

Signals are used for synchronization, but they have flaws:

- First, they can be lost. It is possible for a process to block out signals. If a signal arrives under these conditions, it never will be received. Signals sometimes are blocked for short periods to ensure uninterrupted execution of portions of a program. But the loss of a signal that represents a critical event clearly cannot be tolerated.
- Second, signals cannot be assigned a priority. In most real-time situations, certain events are more critical than others, and must accordingly be assigned a higher priority.
- Finally, only a finite number of signals can be assigned to any one process. Depending on the UNIX implementation, the number of available signals may be too small for the real-time application at hand.

Many implementors of UNIX have "fixed" signals. The Berkeley signal package in 4.2BSD provides a case in point. However, the Berkeley package has problems of its own. In "fixing" signals, the Berkeley developers lost many of the original UNIX signal semantics. They thereby broke a lot of code. Indeed, as Henry Spencer of the University of Toronto so eloquently stated recently: "4.2BSD does everything UNIX does, . . .only differently." [12]

MASSCOMP has taken a different tack on the signals issue. In RTU Version 3.0, four different event packages were implemented: three UNIX signals packages (a "standard" Version 7/System III/System V package, a Berkeley 4.1 package, and a Berkeley 4.2 package) and a new package designed for real-time applications based on the asynchronous system trap (AST).

Since the AST is not closely related to anything in classical UNIX, we describe it under a separate heading.

The Asynchronous System Trap. The concept of an asynchronous system trap, or AST, first appeared in Digital Equipment Corporation's RSX and VMS operating systems [1]. As implemented in MASSCOMP's RTU, an AST is a software interrupt that remedies the defects of signals.

ASTs have programmer-specified priorities, and are queued by the operating system. Delivery of ASTs is guaranteed, and occurs in order of priority. In addition, each AST handler can be passed a parameter when the AST is delivered. Parameters are used to distinguish between different events and often serve as pointers to memory structures.

As with signals, a process that expects to receive a certain AST must set up an AST handler in its process space. The handler is a subroutine called asynchronously during the normal processing of the program. When called, the subroutine runs as part of the user process and thus has access to all operating system capabilities.

ASTs are available systemwide (rather than being open to only a few select processes). The number of ASTs available is determined by the system manager when the system is built.

Predictable Transfer of Data to Backing Store. Real-time data acquisition systems often require more throughput to disk than is possible under standard UNIX. Most UNIX system implementations, even those that use the Berkeley *fast file system*, must map large sections of data to different parts of the disk. The UNIX system disk-buffering scheme generally does an excellent job of caching file-mapping blocks. This means that the overhead of file-mapping seldom involves an additional physical disk read. Even so, its file system overhead may be too great for some real-time applications.

In such cases, one traditional approach has been to implement "contiguous files" by allocating contiguous disk sectors to a single file. This has the net effect of improving file system throughput. Beyond this, it is desirable to reduce all overheads to a minimum so as to write data to disk at the highest speeds possible. One approach is to implement a high-speed data-to-disk mode that grants a single process exclusive access to the disk. In this mode, the system locks out all other accesses to disk, thus eliminating latency for *any* additional seeks.

Many UNIX implementations support "raw disk partitions" that can be exploited in much the same way that contiguous files are. Contiguous files, though, generally provide a *more flexible* way to allocate contiguous disk blocks than does the partitioning of a physical disk at system generation time.

The substitution of a bit map for the UNIX free list is another simple efficiency that has appeared in real-time system implementations (as well as a few timeshared UNIX versions). On most UNIX file systems, the free list [10] points to free blocks that can be written by user processes. Searching this list, though, is a slow task; a bit map containing the same information can be inspected more quickly. A bit map is necessary when contiguous files are implemented because it represents the easiest way to find large contiguous space. An interesting side effect is that a "free-block bit map" uses less disk space than the standard UNIX free list.

Another major change that can be made to the "standard" UNIX file system makes logical disk block size a function of each mounted file system rather than a built-in system parameter. This allows for increased file throughput by using a relatively large block size on large disks (at the cost of some space lost due to fragmentation), while using a smaller block size on floppy disks, where space utilization is more critical.

The Missing Two Requirements. An earlier comparison made between UNIX features and our list of seven real-time requirements showed only two items missing—the synchronization of processes with external events, and the predictable, "instantaneous" response to external events. As it turns out, these two features naturally fall out of work providing the other five features. In MASSCOMP's RTU, for example, external events can generate

Continued to Page 101

C ADVISOR

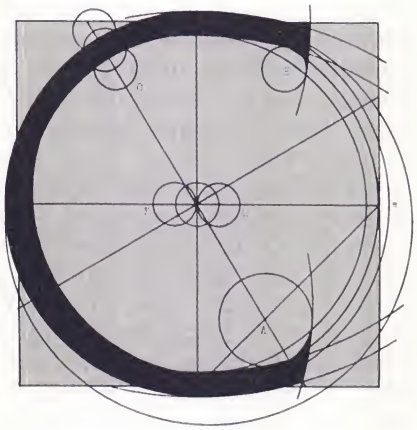
A routine check

by Bill Freiboth and Bill Tuthill

How can a piece of code that is an order of magnitude too large be considered reliable? There is that much more that must be understood in order to make changes. Library functions . . .are one way to reduce the apparent complexity of a program; they help to keep program size manageable, and they let you build on the work of others, instead of starting from scratch each time.

Kernighan and Plauger,
The Elements of Programming Style

Anyone who peruses old UNIX code is shocked by the sheer number of instances where programmers obviously have coded their own functions in preference to using common library routines. Especially prevalent are private string-handling routines which are analogous to **strcpy()** and **strcmp()**. One reason for this is that many UNIX programs were written before such library routines became available. Another is that the UNIX system continues to grow, making it difficult to keep up with all the new library functions that appear. To counter this trend, we offer a comparison of the library routines available on Version 7, System V, and 4.2BSD.



Several years from now, we will look back on today's UNIX software with a jaundiced eye. No doubt, we will be surprised to see custom binary search algorithms, tree management routines, and record locking schemes despite the fact that perfectly good library routines already exist for these purposes. The *System V Interface Definition*, for example, includes specifications for each of these routines.

There are good reasons to use library routines in preference to "rolling your own". First, programs are easier to understand and maintain when other programmers are already familiar with—and can trust—calls to library routines. Second, library routines are often faster than hand-coded functions because of library optimizations. Some vendors, for example, deliver vastly improved versions of **malloc()** and the Standard I/O Library. Third, library routines are maintained by other people, and changes can be coordinated with those made to other parts of the system—meaning that you effectively have use of a maintenence staff not on your payroll. Finally, library routines are documented, and actually do get improved. While documentation may be imperfect, it *is* read frequently, and is likely to be improved in future releases. Private versions of functions tend to be undocumented and hidden in specific programs, waiting to surprise users whenever new releases of the operating system are installed.

As is evident from the table accompanying this article, System V has the most library routines, with 4.2BSD placing a distant second. More than half of the System V additions were actually part of System III, but they are credited to System V since many users did not install System III. The **curses** screen handling functions, and the **termlib** or **termcap** library routines are listed as one-line items. Both have been discussed in this column before. Future columns will discuss library routines available only on System V or 4.2BSD.

The C compilers on System V

Continued to Page 73

Availability of Library Routines

Routine	V7	Sys V	4.2	Comment
C Library				
a64l()		x		convert base 64 ASCII string to long integer
abort()	x	x	x	create a program fault
asctime()	x	x	x	convert a time zone data structure to string format
atof()	x	x	x	convert character string to floating point
atoi()	x	x	x	convert character string to integer
atol()	x	x	x	convert character string to long integer
bcmp()			x	compare a byte array
bcopy()			x	copy a byte array
bzero()			x	zero a byte array
bsearch()		x		binary table search routine
calloc()	x	x	x	allocate an initialized array space
clock()		x		obtain process CPU time
crypt()	x	x	x	encrypt a password using **setkey** and **encrypt** functions
ctime()	x	x	x	convert date and time to string format
ecvt()	x	x	x	convert floating point to string format
encrypt()	x	x	x	encrypt a key using the DES algorithm
endgrent()	x	x	x	end group file processing
endpwent()	x	x	x	end password file processing
endutent()		x		end processing of the accounting file
fcvt()	x	x	x	convert floating point to Fortan F string format
free()	x	x	x	free an allocated storage block
frexp()	x	x	x	split a number into mantissa and exponent
ftok()		x		construct access key for IPC using **msgget**, **semget**, and **shmget** facilities
ftw()		x		descend a directory hierarchy and apply a user-supplied function to each node
gcvt()	x	x	x	convert floating point to Fortran F or E string format
getcwd()		x		obtain the current directory name in string format
getenv()	x	x	x	obtain values for process environment variables
getgrent()	x	x	x	read group file entries sequentially
getgrgid()	x	x	x	read group file entries by group ID
getgrnam()	x	x	x	read group file entries by group name
getlogin()	x	x	x	obtain a pointer to a user login name entry
getopt()		x		obtain command line options
getpass()	x	x	x	read a password from a terminal without echoing
getpw()	x	x	x	obtain a user name from a user ID
getpwent()	x	x	x	read password file entries sequentially
getpwnam()	x	x	x	read password file entries by group name
getpwuid()	x	x	x	read password file entries by group ID
getutent()		x		read accounting file entries sequentially
getutid()		x		search an accounting file by type
getutline()		x		search an accounting file by device
gmtime()	x	x	x	obtain a time data structure containing the GMT time
gsignal()		x		send a signal to a process or a group of processes
hcreate()		x		create a hash-table
hdistroy()		x		remove a hash-table
hsearch()		x		search for an entry in a hash-table
initgroups()			x	initialize group access list
irand48()		x		return double precision random numbers from 0.0 to 1.0
isalnum()	x	x	x	test for alphanumeric character
isalpha()	x	x	x	test for alphabetic character
isascii()	x	x	x	test for ASCII character
isatty()	x	x	x	test whether a file is associated with a terminal

Availability of Library Routines

Routine	V7	Sys V	4.2	Comment
C Library (continued)				
iscntrl()	x	x	x	test for control character
isdigit()	x	x	x	test for digit character
isgraph()		x		test for printable character excluding spaces
islower()	x	x	x	test for lower case character
isprint()	x	x	x	test for printable character
ispunct()	x	x	x	test for punctuation character
isspace()	x	x	x	test for white space character
isupper()	x	x	x	test for upper case character
isxdigit()		x		test for hexadecimal format data
jrand48()		x		return long integer random numbers from -2^{31} to 2^{31}
krand48()		x		return double precision random numbers from 0.0 to 1.0
l3tol()	x	x	x	convert from 3 byte integers to long integers
l64a()		x		convert long integer to base 64 ASCII string
ldexp()	x	x	x	combine mantissa and exponent
localtime()	x	x	x	obtain a time data structure adjusted for local time
longjmp()	x	x	x	restore stack environment information
lrand48()		x		return long integer random numbers from 0 to 2^{31}
lsearch()		x		linear table search and update routine
lto13()	x	x	x	convert from long integers to 3 byte integers
malloc()	x	x	x	allocate a storage block
memccpy()		x		copy memory stopping after a specified character
memchr()		x		search memory for characters
memcmp()		x		compare memory locations lexicographically
memcpy()		x		copy memory to memory
memset()		x		initialize memory to a constant value
mktemp()	x	x	x	make a unique file name using a template
modf()	x	x	x	split mantissa into integer and fraction
monitor()	x	x	x	prepare execution profile for a program
mrand48()		x		return long integer random numbers from -2^{31} to 2^{31}
nlist()	x	x	x	get entries from an executable file's symbol table
nrand48()		x		return long integer random numbers from 0 to 2^{31}
perror()	x	x	x	produce error messages using standard output
pkopen()	x			packet driver simulator
putpwent()		x		write a password file entry
pututline()		x		write accounting file entries
qsort()	x	x	x	quicker sort algorithm
rand()	x	x	x	obtain successive pseudo random numbers range (0,32767)
random()			x	better random number generator than rand()
realloc()	x	x	x	change the size of a storage block
setgrent()	x	x	x	reposition to the start of the group file
setjmp()	x	x	x	save stack environment information
setkey()	x	x	x	initialize a key for use in encryption
setpwent()	x	x	x	reposition to the start of the password file
setutent()		x		reposition to the start of an accounting file
sleep()	x	x	x	suspend process execution for an interval of time
srand()	x	x	x	reset random number generator at a random starting point
swab()	x	x	x	exchange adjacent bytes
tdelete()		x		remove a binary tree node
timezone()	x		x	get the name of the timezone

Availability of Library Routines

Routine	V7	Sys V	4.2	Comment
C Library (continued)				
toascii()		x		convert integer values to ASCII
tolower()	x	x	x	translate characters to lower case (function in Sys V)
toupper()	x	x	x	translate characters to upper case (function in Sys V)
_tolower()		x		macro version of the tolower function
_toupper()		x		macro version of the toupper function
tsearch()		x		create and search a binary tree
ttyname()	x	x	x	obtain the file name of a terminal in string format
ttyslot()	x	x	x	locate the accounting file entry for a terminal user
twalk()		x		traverse (walk) through nodes of a binary tree
tzset()		x		set time zone variables using an environment variable
utmpname()		x		specify the accounting file to be examined
String Functions				
index()	x		x	search for occurrence of character
rindex()	x		x	search backwards for occurrence of character
strcat()	x	x	x	concatenate two full strings
strchr()		x		search for occurrence of character, like **index( )**
strcmp()	x	x	x	lexical comparison of two full strings
strcpy()	x	x	x	copy a string into a second string
strlen()	x	x	x	obtain the length of a string
strncat()	x	x	x	append up to "n" characters to a string
strncmp()	x	x	x	lexical comparison of no more than "n" characters
strncpy()	x	x	x	copy "n" characters of a string
strnspn()		x		obtain length of initial string consisting of characters excluded from a second string
strpbrk()		x		search for a member of a set of characters
strrchr()		x		search backwards for occurrence of character, like **rindex( )**
strspn()		x		obtain length of initial string consisting of characters from a second string
strtok()		x		search a string one token at a time
Standard I/O Library				
clearerr()	x	x	x	reset error, end of file indicators
ctermid()		x		obtain the filename for a terminal
cuserid()		x		obtain the login name of user as a string
fclose()	x	x	x	close a data stream
fdopen()	x	x	x	connect a data stream to an open file
feof()	x	x	x	test for an end-of-file condition
ferror()	x	x	x	test for error conditions
fflush()	x	x	x	flush a data stream without closing it
fgetc()	x	x	x	read a character from an input data stream
fgets()	x	x	x	read a string, but no more than "n" characters
fileno()	x	x	x	obtain the file descriptor for a data stream
fopen()	x	x	x	open a data stream
fprintf()	x	x	x	place output in a named output stream
fputc()	x	x	x	write a character on a data stream
fputs()	x	x	x	write a string onto an output stream
fread()	x	x	x	read buffered input from a data stream
freopen()	x	x	x	redirect output of an open data stream
fscanf()	x	x	x	scan input data from a named input stream
fseek()	x	x	x	reposition random read/write pointer

Availability of Library Routines

Routine	V7	Sys V	4.2	Comment
Standard I/O Library (continued)				
ftell()	x	x	x	determine current position in a data stream
fwrite()	x	x	x	write buffered output to a data stream
getc()	x	x	x	read a character (macro version of **fgetc**)
getchar()	x	x	x	read a character from the standard input (macro)
gets()	x	x	x	read a string up to a newline
getw()	x	x	x	read a word from an input stream
pclose()	x	x	x	close an interprocess data stream
popen()	x	x	x	open an interprocess data stream
printf()	x	x	x	place output in the standard output
putc()	x	x	x	write a character (macro version of **fputc**)
putchar()	x	x	x	write a character to the standard output (macro)
puts()	x	x	x	write a string and append a newline
putw()	x	x	x	write a word on an output stream
rewind()	x	x	x	reposition to the beginning of a data stream
scanf()	x	x	x	scan input data from the standard input
setbuf()	x	x	x	assign a buffer to a data stream
sprintf()	x	x	x	place output in a character stream
sscanf()	x	x	x	scan input data from a character string
ssignal()		x		specify action to perform upon receipt of a signal
system()	x	x	x	issue a shell command
tempnam()		x		obtain filename for temporary file in any directory
tmpfile()		x		create a temporary file
tmpnam()		x		obtain filename for temporary file in */tmp*
ungetc()	x	x	x	put a character back into the input data stream
Math Library				
hypot()	x	x	x	Euclidean distance
acos()	x	x	x	arccosine function
asin()	x	x	x	arcsine function
ceil()	x	x	x	ceiling function
log10()	x	x	x	common logarithm
cos()	x	x	x	cosine function
exp()	x	x	x	exponential
floor()	x	x	x	floor function
cosh()	x	x	x	hyperbolic cosine function
sinh()	x	x	x	hyperbolic sine function
tanh()	x	x	x	hyperbolic tangent function
fabs()	x	x	x	floating point absolute value
abs()	x	x	x	integer absolute value
matherr()		x		math library error handling function
log()	x	x	x	natural logarithm function
pow()	x	x	x	raise a value to a given power
sin()	x	x	x	sine function
sqrt()	x	x	x	square root
tan()	x	x	x	tangent function
gamma()		x	x	log gamma function
fmod()		x		remainder function
erfc()		x		complementary error function : 1 - erf(x)
erf()		x		error function : erf(x)
atan()	x	x	x	arctangent function
atan2()	x	x	x	arctangent function
j[01n]()	x	x	x	Bessel functions of the first kind
y[01n]()	x	x	x	Bessel functions of the second kind

Availability of Library Routines

Routine	V7	Sys V	4.2	Comment
Miscellaneous Routines				
arc()	x	x	x	draw arc given the center and end points
assert()	x	x	x	debugging macro for embedding diagnostic code
circle()	x	x	x	draw circle given center and radius
closepl()	x	x	x	close a plotting device, writing buffered output
cont()	x	x	x	draw line between current position and second point
curses		x	x	cursor addressing and screen updating library
dbm	x		x	database management subroutines
directory			x	directory operations
erase()	x	x	x	clear the plotting area
label()	x	x	x	supply labels for plotting
line()	x	x	x	connect two data points with a line
linemod()	x	x	x	specify style for connecting lines
logname()		x		obtain the login name of a user
move()	x	x	x	reposition the cursor
mp	x		x	multiple precision integer arithmetic library
openpl()	x	x	x	prepare plotting device to receive data
point()	x	x	x	plot a data point
regcmp()		x		compile a regular expression
regex()		x		execute regular expression for a pattern match
re_comp()			x	compile a regular expression
re_exec()			x	execute regular expression for a pattern match
space()	x	x	x	define the perimeter of a plotting space
termlib		x	x	terminal-independent operation library
varargs			x	variable argument list
Network Routines				
byteorder()			x	convert values between host and network byte order
gethostent()			x	get network host entry
getnetent()			x	get network entry
getprotoent()			x	get protocol entry
getservent()			x	get service entry
inet_addr()			x	Internet address manipulation
rcmd()			x	return stream to remote command (superuser)
rexec()			x	return stream to remote command

Continued from page 68

and 4.2BSD (almost identical versions of **pcc**) have been improved since Version 7. Both compilers now support enumeration data types, non-unique structure member names, and the **void** data type (for functions not returning a value). *Long* program identifiers are supported in 4.2, but were not added to System V until release 2. Both System V and 4.2BSD offer profiled function libraries as an aid for software debugging, which means that profiling is supported at the library function level rather than simply at the user program function level.

When should library routines either found only on System V or 4.2BSD be used? And when should you restrict yourself to library routines found on every version of UNIX? If you already have application software on the market, you'll probably be forced to support all versions of UNIX. But if you're writing software now that will reach the market in a year or so, you'll probably find System V compatibility almost everywhere by then. If you need the networking capability of 4.2BSD, then use the Berkeley system because it will probably find its way onto System V before long.

Bill Freiboth is President of Pacific Micro Tech, an El Cerrito, CA, firm engaged in system integration, publishing, and consulting. He formerly served as Vice President of R&D at Decimus Corporation, where he helped design vertical market applications for financial institutions and assess IBM equipment for leasing.

Bill Tuthill was a leading UNIX and C consultant at UC Berkeley for four years prior to becoming a member of the technical staff at Sun Microsystems. He enjoys a solid reputation in the UNIX community earned as part of the Berkeley team that enhanced Version 7 (4.0, 4.1, and 4.2BSD). ■

INDUSTRY INSIDER

What is Smalltalk anyway?

by Mark G. Sobell

Smalltalk has been a big conversation piece ever since its introduction over a decade ago. Despite this, many people still are very confused about what Smalltalk is and what significance it has for the computing community. To help clarify, I recently visited with Dan Ingalls, one of the system's original designers from the Xerox Palo Alto Research Center (Xerox PARC), who—like many of his compatriots—has since migrated to Apple Computer. After we talked for a while in his office at Apple, he switched on his Macintosh to give me a preview of what the company's new Smalltalk environment will look like.

He called up a few windows and menus and showed how the scroll bar allows one to move around in a document, but all the while I was getting fidgety waiting for the Smalltalk demo to start. After all, I already knew how to use a Mac. So when Ingalls started to demonstrate how to use what I took to be MacWrite, I finally asked, "Dan, when are we going to get into Smalltalk?" He looked at me with a puzzled expression and replied, "This *is* Smalltalk."

As it turns out, many of the concepts the Macintosh uses come from Smalltalk. In 1979, Xerox enlisted the support of Apple, DEC, HP, and Tektronix to

help it debug Smalltalk. One of Smalltalk's primary goals was to be a portable environment, so Xerox decided that one of the best ways to debug portability was to let several vendors try to bring up the system on a variety of machines. As part of this process, Xerox gave Apple certain rights to Smalltalk after Apple had succeeded in exorcising the system to Xerox's satisfaction.

A while later Apple came out with the Macintosh (not to mention the Lisa) which, although not a Smalltalk machine, certainly emulated the Smalltalk environment. A number of the user interface techniques originated at Xerox PARC (overlapping windows, mice, pop-up menus, scroll bars) have since appeared on a number of other systems, notably some of the graphics-oriented UNIX systems.

SMALLTALK'S JARGON

Smalltalk, the brainchild of Alan Kay, has its roots in Simula and message passing. According to Ingalls, "Alan often takes an extreme point of view—that's what makes him a good visionary. He took the message-sending model of procedure invocation to its logical extreme in designing Smalltalk." In so doing, he created a world of objects. Each object is an *instance* of a *class*; the class describes the format and also the behavior of all its instances. When anything is done in Smalltalk, it happens because an *object* receives a *message*. The statement:

```
days + 4
```

thus actually reads as, "send the message '+4' to the object **days**". Exactly what is done with the message depends on how the receiver, in this case **days**, interprets the +. The code that **days** runs in response to the message "+4" is called a *method*. Finally, classes are arranged in an inheritance hierarchy so that user-defined classes typically inherit many of their useful properties from pre-existing classes in the Smalltalk system.

Although the individual concepts seem simple enough, the

COMPLETE YOUR UNIX REVIEW LIBRARY!

June/July 1983—UNIX on the IBM/PC . . . . ☐
August/September 1983—Sritek and Venix . ☐
October/November 1983—UNIX Typesetting ☐
December/January 1984—Vi and Emacs . . . ☐
February/March 1984—UNIX Databases . . . ☐
April/May 1984—Menu-based User Interfaces ☐
June 1984—Big Blue UNIX . . . . . ☐
July 1984—The AT&T Family . . . . . ☐
August 1984—Documentation . . . . . ☐
September 1984—System Administration . . . ☐
October 1984—UNIX on Big Iron . . . . . ☐
November 1984—User Friendly UNIX . . . . . ☐
December 1984—Low Cost UNIX . . . . . ☐
January 1985—Evolution of UNIX . . . . . ☐
February 1985—UNIX Portability . . . . . ☐
March 1985—Performance . . . . . ☐
April 1985—UNIX Networking . . . . . ☐
May 1985—Distributed Resource Sharing . . . ☐
June 1985—UNIX Applications . . . . . ☐
July 1985—Office Automation . . . . . ☐
August 1985—Database Intricacies . . . . . ☐
September 1985—Languages . . . . . ☐
October 1985—UNIX and Universities . . . . . ☐

Back issues are $4.95 each including postage. Payment in advance is required. Send this order form with check (US funds payable at US bank only) or credit card information to: REVIEW Publications, 901 S. 3rd St., Renton, WA 98055. Additional $1.00/issue for foreign mail.

Name _____
Company _____
Address _____
City _____ State ___ Zip _____
M/C or VISA # _____
Exp. Date _____

whole picture is elusive. In the introduction to their book on Smalltalk, Goldberg and Robson (see the references at the end of this column) explain this phenomenon by saying, "Due to the uniformity with which the object-message orientation is carried out in the system, there are very few new programming concepts to learn in order to understand Smalltalk. These concepts are presented by defining the five words that make up the vocabulary of Smalltalk—object, message, class, instance, and method. These five words are defined in terms of each other, so it is almost as though the reader must know everything before knowing anything."

During the demo I started to realize the advantage of Smalltalk is that it is a complete environment one can structure in any way—and with any look—that one likes. It can respond in whatever manner you wish because beneath the environment lies a very malleable, portable, object-oriented language.

FUTURE SMALLTALK

Until recently, people outside academia had three major problems with Smalltalk. First on the list was cost and availability: the system would only run on a $30,000 Xerox machine that you couldn't buy. Last year, Tektronix introduced Smalltalk on a $15,000 piece of hardware. This year, it has been brought up on the IBM PC/AT, and a pre-release version is available for the 512 KB Macintosh.

The second problem was that Smalltalk ran very slowly on most machines. With the advent of the 68020 and other fast microprocessors, Smalltalk should run at an acceptable rate on less expensive, single-user machines.

The third problem—yet to be solved—is that because Smalltalk is a whole environment, it takes up a great deal of space. You can put only a limited subset of Smalltalk on a 512 KB Mac, and even then you have only enough room left over for a minor program. You really need a 1 MB Mac with a hard disk to bring up a useful version of the full system.

WHAT IS APPLE UP TO NOW?

A product named MacApp (for

EASIER THAN 1-2-3...

BUT DESIGNED FOR LARGER SYSTEMS

It's simple, C-CALC from DSD Corporation is more flexible, has more functions, and is easier to use than the best selling spreadsheet. We made it that way for a very simple reason, you'll get more work done and make better decisions in less time. That's what makes you successful whether you are planning for the future, forecasting trends, or analyzing profits.

The most popular spreadsheets require a great deal of time to get up and running. When we created C-CALC we kept in mind that time is your most important resource. Our On-Line Help facilities, prompts and menus allow even someone with minimal experience to see meaningful results in very little time. Our built-in training procedures let you pace your own learning with tutorial topics that range from basic to advanced. As you become more experienced, C-CALC allows you to bypass prompts and menus to save even more time.

So call DSD Corporation at (206) 822-2252. C-CALC is currently available for: **UNIX,** VMS, RSTS, RSX, IAS, P/OS, AOS, AOS/VS (Data General), IBM CSOS.

C-CALC is a registered trademark of DSD Corporation. **UNIX** is a registered trademark of Bell Labs. P/OS, RSTS and RSX are registered trademarks of Digital Equipment Corporation. AOS and AOS/VS are registered trademarks of Data General Corporation.

DSD CORPORATION
P.O. BOX 2669
KIRKLAND, WA 98033-0712

EFFECTIVE SOFTWARE FOR BUSINESS

Circle No. 299 on Inquiry Card

Macintosh Application) is already into beta test at Apple. MacApp is an extended, object-oriented Pascal language that Apple claims will help software developers cut the amount of time it takes to code an application. One reason for the claim is that MacApp comes complete with a user-interface library. You can use the library to code the user-interface portion of an application quickly, writing the nuts and bolts in Pascal. But MacApp, like Pascal, is a conventional, compiled language: you need to edit your source, compile it, and then link it before you can execute it. This tedious procedure is not good for prototyping applications.

Because Smalltalk is a good prototyping language, Apple is currently converting it to the architecture of MacApp. With this compatibility, developers will have the option of prototyping an application in the friendly Smalltalk environment and then porting it to a more conventional system for increased efficiency and security. If it flies, the MacApp/Smalltalk connection could be inviting to application developers—a prospect that Apple finds very intriguing indeed.

If you have an item appropriate for this column, you can contact Mr. Sobell at 333 Cobalt Way, Suite 106, Sunnyvale, CA 94086.

Mark G. Sobell is the author of the bestselling book, A Practical Guide to the UNIX System (Benjamin/Cummings, 1984) and the new A Practical Guide to UNIX System V (Benjamin/Cummings, 1985). He has been working with UNIX for over five years and specializes in documentation consulting and ***troff*** *typesetting. Mr. Sobell also writes, lectures, and offers classes in Advanced Shell Programming and* ***awk****.* ■

For more information about Smalltalk, try to find a copy of the August, 1981, issue of *Byte*, which is almost entirely devoted to the subject. There are also two books written by Smalltalk developers: *Smalltalk-80: The Language and its Implementation* by Goldberg and Robson, and *Smalltalk-80: The Interactive Programming Environment* by Goldberg. Both are published by Addison-Wesley. You can obtain more information on the unsupported pre-MacApp version of Smalltalk by writing to Eileen Crombie, Software Library, Apple Computer, Inc., 20525 Mariani Drive, Cupertino, CA 95014.

140 MB disk/tape subsystem for XENIX

Finally, a complete XENIX subsystem for the AT.

Disk Features
- 30, 40, 55, 72, 118 Megabytes (formatted)
- Combine drives with each other or existing drive
- 25 milliseconds average access time
- Simplified installation
- Necessary file modifications done automatically

Tape Features
- 60 Megabyte 1/4 inch cartridge
- Standard XENIX commands (cpio, tar, dd, etc.)
- Fully integrated driver software

Subsystem Features
- Entire subsystem fits inside the AT
- External version with 6 expansion slots available (pictured)
- One year factory warranty

Emerald
Systems Corporation

Mainframe Storage for Micros

4757 Morena Boulevard
San Diego, CA 92117
(619) 270-1994
Telex 323458 EMERSYS
EasyLink 62853804

Emerald & Mainframe Storage for Micros™ Emerald Systems Corp.

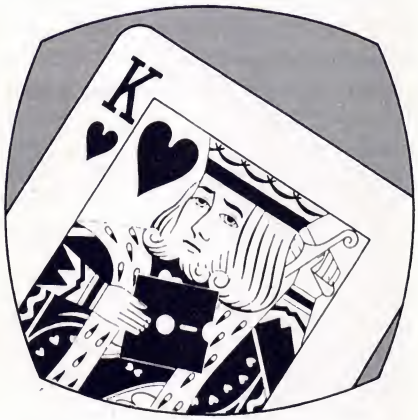
RULES OF THE GAME

Monkey business

by Glenn Groenewold

Readers who have been waiting on the edge of their seats for further developments in the saga of Jack Megabyte need wait no longer. When we last reported on this drama in the October, 1984, column, Jack was being hassled by his former employer, Goliath Corporation, because—along with Helga Termcap, another ace systems designer—he had started a business with the intention of developing and marketing a new operating system capable of outperforming Goliath's. The latter was throwing up legal roadblocks against this enterprise, claiming that Goliath—not Jack—was the owner of any new system he had dreamed up while in the company's employ.

When we took leave of Jack and Helga, we intimated that they most likely would have effective weapons of their own in the legal battle with Goliath. An important part of their arsenal for counterattack could be found among the group of legal actions collectively known as *business-related torts*. Given the increasing frequency with which the game of employee musical chairs is played these days, both in American business generally and in the computing industry in particular, these bases for legal actions (there are more than 20 altogether) are seeing more and more use, as are some of the more familiar legal



varieties of claims for injuries.

WHAT IS THIS THING CALLED "TORT"?

Non-lawyers often find the term "tort" somewhat amusing. But it's a venerable concept in law, where it distinguishes a *civil* impropriety from one that is *criminal* in nature. During the early history of our legal system, the two categories were pretty much distinct. In recent years, however, they've become blurred to the point where it's not unusual for a single act to be the basis both for a civil lawsuit and a criminal prosecution.

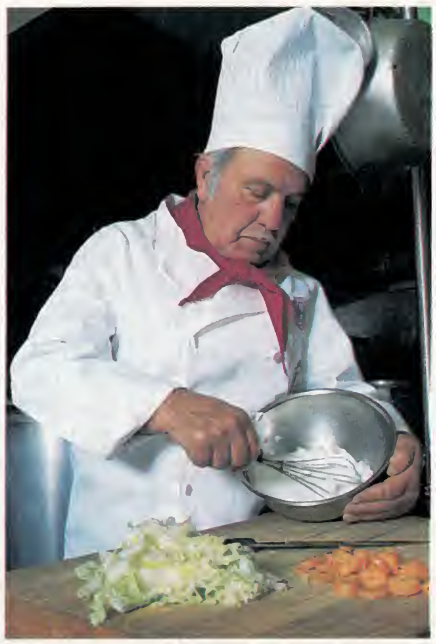
What's more, we even have situations where the same individual may be regarded simultaneously as an "innocent" injured party under one concept and as a malefactor under the other. Thanks to nationwide reportage, many people are aware of the notorious California case involving a would-be burglar who injured himself falling through a skylight on the victim's roof, and then successfully sued on account of his injuries.

Despite this modern-day muddying of the water, the basic notion of a tort remains what it has always been: a *private* wrong, as opposed to a public one. This is why it's such a useful legal concept for individuals who've been injured by someone else's actions.

Some torts, such as *slander* and *libel*, are ancient. Others have developed relatively recently, or have been expanded far beyond their historical scope. The concept of "infliction of emotional distress" is a prime example. In the current sense, this tort scarcely existed in common law, and was almost impossible to prove before its modern evolution.

THAT UNCERTAIN FEELING

Since business-related torts are a matter of state law, not all of them exist in every state. And when the same tort *is* found in various states, it may be known by various names. Some business torts clearly have no application to disputes involving a former employer, an erstwhile employee,



FUSION™ is the only connection.

In an industry fragmented by diversity, FUSION makes connections.

FUSION is the LAN software that links different operating systems, diverse LAN hardware, and diverse protocols, to give you high speed communication across completely unrelated systems.

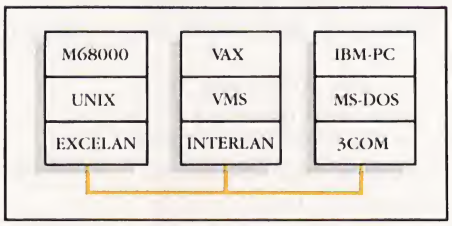
FUSION frees you to buy the best computer for the job—without worrying about compatibility. Your VAX mainframe can now communicate with your IBM-PC. You can even add M68000 work stations or a communications server. FUSION links them all—and more. Regardless of operating system, protocol, or network hardware.

FUSION™

- With FUSION, any user on any system can perform file transfer, remote login, and remote execution.
- FUSION offers a choice of two standard protocols—XNS or TCP/IP. Some customers have even developed private protocols.
- And you can create your own application programs, using FUSION utilities.
- FUSION accommodates all major LAN boards, too. So network interface is no obstacle.

When you add FUSION, you never lose your hardware investment. In fact, you get more out of it than ever.

Ask for FUSION. It's the connection you've been looking for.



A sample network using a few of the many configurations made possible with FUSION.

For further information please contact:
Network Research Corporation
2380 North Rose Avenue
Oxnard, California 93030
(805) 485-2700
(800) 541-9508

FUSION is a licensed trademark of Network Research Corporation.

Circle No. 252 on Inquiry Card

network research corporation

or the latter's new employer or business enterprise. But in a given situation, the relevance of some of these other disputes may be the subject of contention.

Moreover, the legal requirements necessary to prove any of these actions are highly technical, and vary from state to state. As a result, it's not possible here to do more than to suggest remedies that *might* be available to participants in controversies resulting from the ongoing business activities of a former employee.

FROM THE EMPLOYEE'S PERSPECTIVE

Two alternative possibilities exist when a departing employee intends to continue a remunerative activity in potential competition with a former employer. Each presents a different legal scenario. On the one hand, the employee may expect to take a job with another concern in the same field. Or, as with Jack Megabyte, he or she may attempt to launch a competing enterprise. We'll begin by considering the first of these situations.

There are several ways an employer might find itself on the receiving end of a lawsuit brought by a former employee if it seeks to prevent employment by a competitor. For instance, if it responds to an inquiry on the part of the prospective employer with something like, "Well, hire her if you want to, but you ought to know about her $150 a day habit", or perhaps, "We're happy he left before he got a *third* receptionist pregnant", it had better be able to prove these things, lest it find itself stuck with damages for the old-fashioned tort of *slander* (or *defamation*, as it's often called nowadays). Even if the former employer can establish that any personal information of this sort is true, it may nevertheless find itself sued on the basis of *invasion of privacy*.

Assuming the employer does not resort to such canards, but still dissuades a prospective new employer from hiring its ex-employee, it might nevertheless expose itself to a claim of *interference with prospective economic advantage*. For example, suppose it threatens to bring a lawsuit against the prospective employer if it hires the former employee, or to terminate a licensing or distribution agreement.

> We have situations where the same individual may be regarded simultaneously as an "innocent" injured party under one concept and as a malefactor under the other.

Finally, if all else fails, in liberal jurisdictions like California there's the rapidly expanding tort of *infliction of emotional distress* to give redress to a former employee who has been blocked vindictively from obtaining new employment.

Where the employee proposes to follow the alternate route of launching his or her own business and the former employer attempts to thwart this, some of the legal actions just mentioned may be available. However, there are others as well.

The basic business tort applying to this type of situation is *interference with the right to pursue a lawful business*, which is pretty much what its name suggests. In addition, there are less obvious measures that may present themselves in certain cases.

If the former employer has required the employee to disclose all of his or her creations during the employment, and the facts indicate that the material in question actually was created outside the scope of employment—making it the employee's property—a claim of *misappropriation of trade secrets* or *copyright infringement* may be in order. (The latter, since 1978, has been exclusively within the jurisdiction of the federal courts. It therefore no longer constitutes a *state* business tort.)

If none of these actions appears to fit, the situation may permit an action on the basis of *unfair competition*. Originally, in common law, this tort was rather narrow in scope, essentially applying only to situations in which the *buying public* had been misled by some action on the part of a business competitor. In recent years, it has expanded to become rather a catch-all, so that it now can encompass such things as unfair tactics on the part of a former employer designed to prevent competition from its ex-employee.

FROM THE FORMER EMPLOYER'S PERSPECTIVE

The employer, however, is not exactly without legal weapons for use in lawsuits against its estranged employee, and, in some cases, in actions against a new employer who has alienated the

Sorry, Countess, this is one computer where you won't find VADS™!

Although the VERDIX Ada® Development System(VADS) won't be rehosted on Charles Babbage's Difference Engine, it is being hosted on and targeted for a variety of computer systems and embedded system architectures.

The Department of Defense (DoD) has now validated VADS for a growing number of computers and operating systems including the DEC/VAX™ series under UNIX™ 4.2 BSD and ULTRIX™, and for the Sun-2™ Workstation. Future product releases will include Host Development Systems for VAX/VMS™ and UNIX System V, and cross-targeted systems for 4 major architectures... Motorola 68000 and Intel "86" families, the NS32032, and MIL-STD-1750A.

VADS is the fastest and friendliest Ada development system available. It is specifically designed for large-scale Ada program development in a production environment.

VADS features a complete run-time system, plus an interactive, screen-oriented, fully symbolic debugger that lets you easily pinpoint errors. Unexcelled diagnostics and Ada library utilities quickly manage, manipulate and display program library information, dramatically shortening development times.

VADS from VERDIX. The finest, fastest and most cost-effective Ada Development System on the market today. The biggest breakthrough in programming since Ada herself.

For full information, call Jack Crosby, Director of Marketing, at (703) 378-7600.

VERDIX

14130 Sullyfield Circle, Chantilly, VA 22021

Ada is a registered trademark of the U.S. Government, Ada Joint Program Office.
VAX, VMS and ULTRIX are trademarks of the Digital Equipment Corporation
UNIX is a trademark of Bell Laboratories
Sun-2 is a trademark of Sun Microsystems, Inc.
VERDIX and VADS are trademarks of Verdix Corporation

Circle No. 250 on Inquiry Card

employee's affections.

The latter can itself be sued for *misappropriation of trade secrets* if it can be shown that one of the reasons it hired the employee was to obtain access to the trade secrets that he or she had learned in the former employment. The new employer could also be the subject of a claim of *unfair competition* for having lured a key employee away from its competitor, or it could be sued for *inducing a breach of contract*—which in this case would be the employee's contract of employment with the former employer. In some states, the new employer also might be liable for *interference with an employment relationship*, though this tort ordinarily has been applied only when a *physical* injury has occurred to an employee. Thus, from our standpoint, this tort largely would appear to be preempted by the tort of unfair competition.

Turning its attention to its former employee, the erstwhile employer may have legal actions available based on *misappropriation of trade secrets*, *copyright infringement*, or *unfair competition*, all of which have been discussed here. But it may also have a claim based upon the employee's *breach of contract*. This is where the contract of employment between the employer and employee (which we considered last September) becomes highly important.

If, for example, the employee had agreed that he or she would not take employment with a competitor for a specified time after leaving the first employer, or would not start a business in competition with the employer, this provision might be enforceable, depending on all of the circumstances. But because *reasonable* restrictions of this type *may* be valid in whole or in part, employees in key positions should take the precaution of obtaining competent advice at the time the terms of their employment are negotiated.

AND NOW, BACK TO OUR STORY

So what is to happen with Jack Megabyte in his legal battle with Goliath?

I hope the contents of this article are sufficient to illustrate why I can't possibly answer that question with any degree of certainty. In most places, the odds are that the various lawsuits will take what seems like an eternity to go to trial. When they finally are debated in court, the results will depend on the multiplicity of facts that are developed and, of course, the laws of that particular state.

I'd like to think that during the long wait, Jack will manage to strike it rich in the Publishers' Clearinghouse sweepstakes and find time to luxuriate on Majorca, while Helga turns her talents to successfully franchising a chain of *t'ai chi* exercise studios, to her immense profit. We all like happy endings, don't we?

Glenn Groenewold is a California attorney who devotes his time to computer law. He has served as an administrative law judge, has been active in trial and appellate work, and has argued cases before the state Supreme Court. ■

FRANZ
THE FIRST NAME IN
LISP

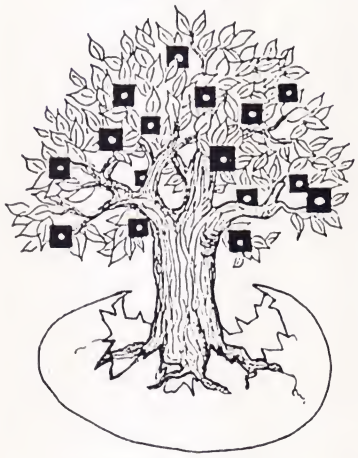
Franz LISP from Franz Inc. is currently available under UNIX and VMS. Now with Flavors and Common LISP compatibility. Franz sets the standard for LISP.

Franz Inc.
1141 Harbor Bay Parkway
Alameda, California 94501
(415) 769-5656

UNIX is a trademark of Bell Labs. VMS is a trademark of Digital Equipment Corporation.

Circle No. 287 on Inquiry Card

Tree Shell
A Graphic Visual Shell for Unix/ Xenix End-Users and Experts Alike!



Dealer inquiries welcomed.
COGITATE
"A Higher Form of Software"
24000 Telegraph Road
Southfield, MI 48034
(313) 352-2345
TELEX: 386581 COGITATE USA

Circle No. 288 on Inquiry Card

CLEO is your SNA or BSC Gateway

Connect your IBM, Apple, Tandy, Zenith, A.T.&T., Hewlett-Packard, Televideo, NCR, IMS, SUN, or other DOS or UNIX-based system to another micro or to your mainframe with CLEO Software.

Now you can connect your PC LAN, too!

For details call: 1(800) 233-CLEO
In Illinois 1(815) 397-8110

CLEO™

CLEO Software
a division of Phone 1, Inc.
1639 North Alpine Road
Rockford, IL 61107
TELEX 703639

Circle No. 265 on Inquiry Card

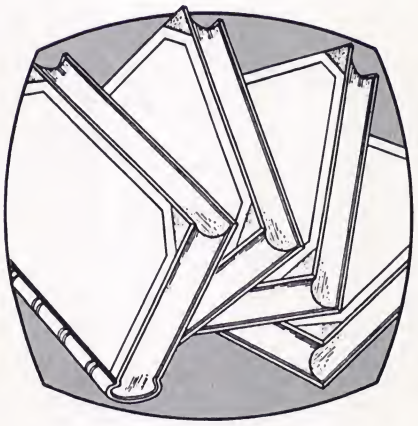
CLEO and 3780Plus are registered trademarks of CLEO Software.
IBM is a registered trademark of International Business Machines Corporation; Apple is a registered trademark of Apple Computer; UNIX is a registered trademark of A.T.&T. Technologies, Inc.

FIT TO PRINT

It's in the stars

by August Mohr

A premise of the *Whole Earth Catalog* holds that the best introduction to a book is the book itself. I wholeheartedly agree, so in launching this new book review column I will attempt to let authors speak for themselves in a way that, I hope, will let you get an accurate idea of what the book is about.



Three "guides" to UNIX have crossed my desk in the past month or so, and for different reasons they all deserve comment. They are: *The UNIX Environment*, by A.N. Walker; *A Practical Guide to UNIX System V*, by Mark G. Sobell; and *XENIX by Example*, by The Staff of M & M Technologies Corporation.

The UNIX Environment

Andrew N. Walker
151+xi pp. ISBN 0-471-90564-X
John Wiley & Sons, 1984
605 3rd Ave.
New York, NY 10158
$15.95 (paper)

Walker has produced a delightful book. As soon as I opened it, he already had captured me with anecdotes. Walker clearly enjoys his subject, and his pleasure is infectious. I read passages to my wife, who knows little about UNIX, and got many honest chuckles from her. Walker's lighthearted style is completely appropriate since the book is intended to communicate what using UNIX is like, rather than *how* it should be used.

It is clear that Walker is writing to the computer professional who is not familiar with UNIX but wants to know more about it without actually having to use a system. In his own words:

> *I hope that by the end of this book, ... if you do not already use Unix, you will be able to persuade your company, or your institution, or your rich uncle, or whatever, to give you a Unix installation to play with. Unless you have an incredibly boring view of the nature of computing, your life will never be the same again.*

My own introduction to UNIX came by way of a 1981 article in *Computer* magazine authored by Kernighan and Mashey called "The Unix Programming Environment". This book is very reminiscent of that article.

Walker also goes beyond introducing readers to the style of UNIX in his discussion of the system's internals. Because he is writing to computer-oriented people, he explains the concept of **fork** and **exec** before he even gets to the shell. A discussion of the kernel's view of the file system comes even earlier.

> *How do we make available a particular block of the disc? Well, we first of all scan the buffer pool to see if the block is already available; if so, we can return immediately. If not, we have to find an empty buffer, issue a request to the hardware, and wait for the information to arrive. How do we find an*

NAME THE MOST WIDELY USED INTEGRATED OFFICE AUTOMATION SOFTWARE FOR UNIX™ SYSTEMS.

"UNIPLEX II"™

YOU'VE GOT IT!

User satisfaction is the primary reason no other product can make this claim. Already in its second generation, UNIPLEX II offers features designed to meet the requirements of the most demanding user.

The beauty of UNIPLEX II is its simplicity. One personality and one command structure throughout the program provide an ease of use never before experienced with UNIX application software.

UNIPLEX II integrates sophisticated word processing, spreadsheet, and relational database applications into a powerful one-product solution.

UNIPLEX II uses termcap, so it can run on virtually any computer terminal. "Softkeys" allow the user to define function keys which are displayed on the 25th line of most terminals to provide versatility and ease of use.

All this at a price you'd normally pay for a single application software package.

UNIPLEX II is available immediately from UniPress Software, the company that's been at the forefront of quality UNIX software products longer than anyone else.

Call today! Once you've got it, you'll see why UNIPLEX II is the most widely used integrated office automation software for UNIX-based systems.

OEM terms available. Mastercard and Visa accepted!

Write to: UniPress Software, 2025 Lincoln Hwy., Edison, NJ 08817 or call: 1-800-222-0550 (outside NJ) or 201-985-8000 (in NJ); Telex: 709418. European Distributor: Modulator SA, Switzerland 41 31 59 22 22, Telex: 911859.

UNIX is a trademark of AT&T Bell Laboratories. Uniplex II is a trademark of Uniplex Integration Systems.

NOW AVAILABLE ON THE AT&T UNIX PC 7300 & 3B SERIES!

UniPress Software

Your Leading Source for UNIX™ Software

Circle No. 260 on Inquiry Card

The best introduction to a book is the book itself.

empty buffer? Well, with luck there is one not in use at all. Failing that, we latch on to a buffer that has not changed from its equivalent on the disc; this can be re-used straight away. Failing that, we find a buffer that is unlikely to be wanted soon, issue a request to the hardware to copy it back to the disc, and then we can grab it for our own purposes. Requests to the hardware are themselves buffered, and possibly dealt with out of order to improve efficiency.

Naturally, real life is much more complicated than its simplified description here. Tables get full, buffers are soon all in use, and stringent precautions must be taken against deadlocks, races, errors, and breaches of security; but such fine detail can safely be left to the reader's imagination.

In keeping with the notion of a computer-literate audience, Walker discusses interprocess communication, **make**, and the C language before getting into the **vi** editor and **nroff**. His discussion of C is excellent, both as an introduction to the flavor of the language and for its comparisons with other languages. His sample programs all have good style. The largest, 22 lines, is a working program written to reformat a tape file intended for another system. His top-down, line-by-line explanation of the program's use of pointers should help make C comprehensible even to people who have only BASIC experience.

The use of { . . . } to bracket compound statements, rather than the more usual **begin . . . end**, *and the elision of* **then** *are typical details that contribute to the rather opaque appearance of C. Many of my students use the macro pre-processor facility to define:*

```
#define IF    if(
#define THEN  ){
#define ELSE  :}else{
#define FI    :}
```

(for example), to replace strings of brackets by more readable keywords. After this they can pretend they are writing Algol 68:

```
IF i > j THEN x = y ELSE printf ("error!") FI
```

I am disappointed that the book is limited to Version 7 UNIX, and that it often makes comparisons with Version 6. This leaves readers to determine for themselves whether this or that problem or feature still exists in System III or System V. But since Walker's book is not intended as a working manual, this is a minor problem.

The book is well edited, as one would expect a John Wiley & Sons product to be. I did not find any obvious errors in the examples, and all of the quote marks were of the proper kind. The only typo I could find was an instance where the name of the root directory was to be displayed on a line by itself, and only a blank line was produced instead. The author's britishisms only occasionally bothered me: "disc" for "disk" I didn't mind, but I stumbled over "transput" for "input/output".

Command lines are universally separated from the text in a display format and set in a clean Helvetica type. The braces, brackets, stars, and

IBM XENIX DISK-TAPE-RAM

FOR THE IBM PC/AT—XENIX® OR DOS:

86 MEGABYTE HARD DISK - $2495
28ms average access, longest MTBF, 1 year warranty. Finest quality drives made.

60 MEGABYTE TAPE BACKUP - $1695
90 IPS, 5MB/minute cartridge tape. We ship the same unit IBM sells. Highest performance.

2 MEGABYTE RAM CARD - $745

120ns RAM, fully populated. Why pay more?

FIND OUT about the IBM Unix® Solution with 86MB Disk, 60MB tape, 2.5MB RAM and better than VAX 750® floating point for under $9500, quantity one price including all software. Why pay three times more for a slower machine?

Bell Technologies
415-794-5908 / PO Box 8323
Fremont, California 94537

Call today for quantity discounts.

Circle No. 286 on Inquiry Card

NEW RELEASE

UNIPRESS EMACS™

EDITOR FOR: UNIX™/ VMS™/ MS-DOS™

Another in a series of productivity notes on software from UniPress.

Subject: Multi-window, full screen editor.

Multi-window, full screen editor provides extraordinary text editing. Several files can be edited simultaneously, giving far greater programming productivity than vi. The built-in MLISP™ programming language provides great extensibility to the editor.

New Features:

- *EMACS is now smaller and faster.*
- *Sun windows with fonts and mouse control are now provided.*
- *Extensive on-line help for all commands.*
- *Overstrike mode option to complement insert mode.*
- *New arithmetic functions and user definable variables.*
- *New manual set, both tutorial and MLISP guide.*
- *Better terminal support, including the option of not using unneeded terminal drivers.*
- *EMACS automatically uses terminal's function and arrow keys from termcap and now handles terminals which use xon/xoff control.*
- *More emulation-TOPS20 for compatibility with other EMACS versions, EDT and simple Wordstar™ emulation.*

Features:

- *Multi-window, full screen editor for a wide range of UNIX, VMS and MS-DOS machines.*
- *"Shell windows" are supported, allowing command execution at anytime during an edit session.*
- *MLISP programming language offers extensibility for making custom editor commands! Keyboard and named macros, too.*
- *"Key bindings" give full freedom for defining keys.*
- *Programming aids for C, Pascal and MLISP: EMACS checks for balanced parenthesis and braces, automatically indents and reformats code as needed. C mode produces template of control flow, in three different C styles.*
- *Available for the VAX™ (UNIX and VMS), a wide range of 68000 machines, AT&T family, Pyramid,™ Gould,™ IBM-PC,™ Rainbow™ 100+ and many more.*

Price:

	Binary	Source
VAX/UNIX		$995
VAX/VMS	$2500	7000
68000/UNIX	395	995
MS-DOS	325	995

*For our **Free Catalogue** and more information on these and other UNIX software products, call or write:*
UniPress Software, Inc.,
2025 Lincoln Hwy.,
Edison, NJ 08817.
Telephone: (201) 985-8000.
***Order Desk: (800) 222-0550 (Outside NJ).** Telex: 709418.*
European Distributor:
Modulator SA, Switzerland
Telephone: 41 31 59 22 22,
Telex: 911859.

OEM terms available.
Mastercard/Visa accepted.

Trademarks of: UniPress EMACS & MLISP, UniPress Software, Inc.; UNIX, AT&T Bell Laboratories; VAX/VMS & Rainbow 100+, Digital Equipment Corp.; MS-DOS, Microsoft Corp.; WordStar, MicroPro; Pyramid, Pyramid; Gould, Gould.

Circle No. 261 on Inquiry Card

UniPress Software

Your Leading Source for UNIX™ Software.

Walker clearly enjoys his subject, and his pleasure is infectious.

pipes are all clear. My only complaint about the layout is that there are no chapter references at the tops of the pages to help readers find their way around. That is acceptable in a book intended for regular reading, but headers should be mandatory in a reference work. Since the author clearly intends this as an informal introduction, this omission is forgivable here, but I like to browse and would have preferred landmarks.

A Practical Guide to UNIX System V

Mark G. Sobell
577+xii pp. ISBN 0-8053-8915-6
The Benjamin/Cummings Publishing Company, Inc., 1985
2727 Sand Hill Road
Menlo Park, CA 94025
$20.95 (paper)

Mark Sobell has updated his excellent book on Version 7 UNIX to include relevant aspects of System V. This book is intended to be both an in-depth tutorial and a reference guide. It succeeds well on both counts.

One of the best aspects of the book is its use of examples to illustrate the actions of different commands, options, and syntaxes. By giving readers the text of simple files and showing the various possible results, Sobell teaches a simple technique for establishing the behavior of a command without recourse to a manual. This may not seem like much, but for a beginner to learn the "try it and find out" approach can be a major threshold. Sobell makes it safe, easy, and alluring.

His section on **vi** is one of the best available. Visual explanations for essentially two-dimensional commands make good sense, especially when they're well done—as they are in this book.

However, despite the fact that Sobell's new book is a revision, occasional errors have seeped through. For instance, he writes, "The **b** key moves the cursor backward to first letter of the previous word." The accompanying picture shows the cursor moving back from the middle of the current word to the beginning of the previous word. That is inaccurate. A better phrasing would have indicated that the cursor would move "to the previous first letter of a word." The cursor should then be shown moving to the beginning of the current word.

I hope such nitpicking does not give a false impression. This is a well-done book.

Xenix by Example

The Staff of M & M Technologies Corporation
M & M Technologies Corporation, 1984
PO Box 237 Herndon, PA 17830
$39.95 (paper)

This is a disappointing book. It is marred by poor editing and inaccurate examples. These problems are compounded by the fact that the book is also ugly. The index and table of contents were produced on a line printer, as were the diagrams—even the pictures of directory tree structures. The pages are numbered by chapter, making it difficult to locate oneself in the book.

The book's most redeeming feature is a nice section at the end telling of a week in the life of a system administrator. This piece actually serves as a useful description of the job.

Because of the book's orientation towards Xenix on Tandy systems, I suspect it is intended to be distributed as part of a package. In that form it may have some usefulness, but I would not recommend buying it separately for its own merits. It is not a *bad* book, but it could easily have been better.

Xenix by Example is marred by poor editing and inaccurate examples.

It's in the stars: 0 = kitty litter; 1 = take it if it's free; 2 = worth the cover price; 3 = well worth reading; 4 = get the leather-bound edition.

August Mohr is the new book review editor for UNIX REVIEW. With a background in both computer science and publishing, Mr. Mohr has combined these interests while working with the international UNIX users' organization /usr/group. He was the founding editor of the newsletter/magazine CommUNIXations, and also served as the compiler and producer of the group's UNIX Products Catalog. ■

AN EXTENSIVE LINE OF PROGRAMMING LANGUAGES FOR NATIVE AND CROSS DEVELOPMENT WORK

SVS™ FORTRAN 77/ Pascal/BASIC-PLUS/C

SVS family of native mode compilers for MC68000™ UNIX™ machines. Full ANSI standard, symbolic debugger and optimized code generator with high speed optimization. Support for IEEE floating point, both single and double precision. SVS languages give excellent performance.

u4th

FORTH programming language for UNIX. Largely compatible with the FORTH-83 standard. u4th is interactive and allows full UNIX system call interface, as well as UNIX command pass-through. Permits C primitives and FORTH words to be loaded into a new kernel image. Used frequently in Artificial Intelligence work.

SSI TOOLKIT

SSI Toolkit is a set of Intel-style cross development tools for UNIX and VMS.™ Package includes macro cross assembler compatible with Intel ASM-86/87/88/186/188, linker, locator and librarian.

LATTICE® C CROSS COMPILER

Use your VAX™ (UNIX or VMS) or other UNIX machine to create standard MS-DOS™ object code for 8086™ and 186.™ The Lattice package includes compiler, linker, librarian, disassembler and 8087™ floating point support. Optional SSI Intel-style cross development tools can be used in conjunction with Lattice for native mode 8086 applications.

AMSTERDAM COMPILER KIT

Now includes BASIC, additional back-ends and simpler licensing!

A package of C, Pascal and BASIC (native and cross) compilers for UNIX machines. Hosted on VAX, PDP-11,™ MC68000,™ Z8000™ and 8086. Targets for VAX, PDP-11, MC68000,™ 6500™/6502,™ Z8000, 8086, NSC16032™ and 8080/Z80.™ Cross assemblers provided for MC6800™/6805™/6809,™ NSC16032 and Signetics 2650.™ Package contains complete sources.

PRICE:

SVS Languages	
FORTRAN 77, Pascal, and C	$995 each
BASIC-PLUS	750
u4th	
IBM-PC AT,™ Intel 286/310™	195
MC68000	495
VAX	1500
SSI Toolkit	
VAX (UNIX and VMS)	8000
MC68000	5000
Lattice C Cross Compiler	
VAX (UNIX and VMS)	5000
MC68000	3000
Amsterdam Compiler Kit	
Commercial users	9950
Educational Institutions	995

For our free catalogue and more information on these and other UNIX software products, call or write:
UniPress Software, Inc., 2025 Lincoln Hwy., Edison, NJ 08817.
Telephone: (201) 985-8000.
Order Desk: (800) 222-0550 (Outside NJ).
Telex: 709418.
European Distributor: Modulator SA, Switzerland. Telephone: 41 31 59 22 22, Telex: 911859

OEM terms available. Mastercard and Visa accepted.

Trademarks of: UNIX, AT&T Bell Laboratories, VAX, VMS, and PDP-11, Digital Equipment Corp.; 8086/8087/186/286/310, Intel Corp.; MS-DOS, Microsoft, Lattice, Lattice Inc.; MC6000, 6500, 6502, 6800, 6805, 6809, Motorola Corp.; NSC16032, National Semiconductor Corp.; Signetics 2650, Signetics Corp.; SVS, Silicon Valley Software, Inc.

UniPress Software
Your Leading Source for UNIX Software.

Circle No. 258 on Inquiry Card

DEVIL'S ADVOCATE

Setting the sights

by Stan Kelly-Bootle

I have been asked to keep you updated on my exciting chase after Ty Cobb's lifetime hit total. My attempt will obviously challenge Pete Rose's record also, but that is entirely coincidental. Relax, Peter my dear boy, it's Tyrant R. Cobb I'm after.

For some time I've considered tackling Cobb's achievements in non-Wrigley-type fields, off-the-plate as it were. Take Ty Cobb's record in, say, MC68000 assembly language programming. You needn't look it up; Cobb's coding was unbelievably ineffective at all levels, pure Bugsville—0 for FFFF, man. There's no fun in beating such an abysmal performance.

No, the improbable dream should be made of sterner stuff, as Hamlet once remarked to Brutus. Why drive up Mount Tamalpais when there are the Eigers to be scaled? Why grovel in RPG when the perfect payroll cries out for Lisp? You can test the eerie intelligence of Lisp by typing:

```
(GET 'STAN 'RAISE)
```

which sadly but correctly returns: NIL.

So here is the challenge I have taken on amidst a sea of doubting sniggers:

```
Ty Cobb's record:
  Total Hits ____________________4191
```

```
Stan ''Mr. November'' Kelly-Bootle:
  Last Game ____________________0-5
  Total Hits ___________________0
  Hits still needed to tie Ty ________4191
  Hits still needed to pass Ty _______4192
  Hits still needed
    to completely humiliate Ty _____8192
```

You will be relieved to learn that I plan to retire at this very point, just as my 11-bit register overflows!

Be sure to follow my progress, reported exclusively here in UNIX REVIEW every month; you will definitely *not* find it in the so-called Sporting Press. (I leave you to draw your own conclusions; just remember that Pete Rose's first at-bat also attracted very little attention from these same smart-alec jockscribes.)

I am not the only Quixote on the block! Indeed, I am much encouraged by letters from Dorothy D'Attoma, the lively PR person for Multi Solutions Inc., creators of the S1 operating system. I have a theory that S1 stands for Sisyphian #1, indicating that Dorothy has one hell of an uphill assignment, namely dislodging UNIX from the top of the heap.

I recall selling Univac's Exec8 against IBM's OS360 in those carefree days of core. (By the way, whatever happened to plated-wire memory? Sure, it was three bucks-a-byte, but *very* pretty.) If we were asked if Exec8 had a certain feature or property, we would ponder to ourselves if such a feature or property seems sensible and desirable. If so, our answer would be an enthusiastic affirmative: "Yes, sir. What's more, it's transparent to the user." We would then rush off a cable to Sperry HQ in Bluebell, PA pleading for enhancements.

Another historical curiosity springs to mind, although it predates my own personal involvement. During the 1840s, the British steam locomotive rail system had evolved haphazardly with private companies building networks with different track widths. Each, of course, had convincing arguments why a 5-foot 6-3/4 inch gauge was either a God-given boon or a pitiful, diabolical trap. Certainly those with trains and rolling-stock running

on wheels 5-foot 6-3/4 inches apart had difficulty admitting the sanity of anyone who sang the praises of a 7-foot 1-inch wide track.

Nevertheless, the greatest engineer of the day, Isambard Kingdom Brunel (1806-59), proved beyond doubt that the wider track offered a safer, more comfortable ride. Further, he went ahead and built the Great Western Railway on this basis. For a few years, trains were actually designed with adjustable wheels, allowing them to make the journey from, say, Norwich to Bristol with two or three stops for wheel adjustment (preferable to the alternative "All Change!" for goods and passengers). Alas, poor Brunel was just a little late; the narrow gauge, devised before all the design implications were known, became the prevalent, unshakeable *standard*.

This raises the obvious question (if you like simplistic analogies): is UNIX running on the right lines for the wrong reasons? Or vice versa? Who's on first? Not me, I'm still looking for the first of those damned elusive 4191 hits.

Is UNIX running on the right lines for the wrong reasons?

Newsflash: T. Boone *Pick* has taken over AT&T and the University of California at Berkeley. His operating system has been declared the ad hocissimus standard. *All Change!*

Damn, now where did I put my old BASIC manuals?

Liverpool-born Stan Kelly-Bootle has been computing, on and off, at most levels since the pioneering EDSAC I days in the early 1950s at Cambridge University. After graduating from there in Pure Mathematics, he gained the world's first post-graduate diploma in Computer Science. He has authored The Devil's DP Dictionary and co-authored Lern Yerself Scouse and The MC68000 Software Primer. ■

Another in a series of productivity notes on UNIX™ software from UniPress.

SPREADSHEET

Q-CALC

Subject: Powerful spreadsheet with NEW ADDED FEATURES.

Q-Calc is an extraordinary spreadsheet for UNIX including extensive math and logic facilities, comprehensive command set, optional graphics, many new ease-of-use features, and the ability to run UNIX programs on spreadsheet data.

Features:

- *Fast spreadsheet with large model size, allowing sorting and searching.*
- *Interfaces with UNIX and user programs via pipes, filters and sub-processes. Data can be processed interactively by UNIX.*
- *Q-Calc profile mechanism allows the user to store default information, as well as support for terminal-specific profiles. Uses termcap.*
- *Graphics for bar and pie charts. Several device drivers supported.*
- *New Features of Version 3.2 include more powerful printing, simpler data input, keybinding definitions, new string operator, bind-to-key, and more.*
- *Available for the VAX™, Sun™, Masscomp™, AT&T 3B & 7300 Series, Pyramid™, Plexus™, Gould™, Cadmus™, Integrated Solutions™, Cyb™, IRIS™, Callan™, and many more.*

Price:

	Binary
VAX, Pyramid, AT&T 3B/20	$2500
(with graphics)	3500
MC68000™	750
(with graphics)	995

Source Code Available.

*For our **Free Catalogue** and more information on these and other UNIX software products, call or write:*
UniPress Software, Inc.,
2025 Lincoln Hwy., Edison, NJ 08817.
Telephone: (201) 985-8000.
Order Desk: (800) 222-0550
(Outside NJ). *Telex: 709418.*
European Distributor: Modulator SA, Switzerland Telephone: 41 31 59 22 22, Telex: 911859.

OEM terms available.
Mastercard/Visa accepted.

Trademarks of: UNIX, AT&T Bell Laboratories; VAX, Digital Equipment Corp; Sun, Sun Microsystems; Masscomp, Masscomp; CYB, CYB Systems; Plexus, Plexus Computer; Gould, Gould; Pyramid, Pyramid; Integrated Solutions, Integrated Solutions; IRIS, Silicon Graphics; Cadmus, Cadmus Computer; Callan, Callan Data Systems; MC68000, Motorola Corp.

UniPress Software
Your Leading Source for UNIX™ Software.

Circle No. 281 on Inquiry Card

THE UNIX GLOSSARY

Real-time vernacular

by Steve Rosenthal

Note: only the meanings most pertinent to scientific and real-time programming have been included in this listing.

adaptive control—a regulation that alters the mode, speed, or type of system response according to changing circumstances or results. The use of adaptive controls is desirable in many kinds of real-time control systems, particularly robotics.

AP—an abbreviation for "*array processor*" (see below) or "*attached processor*" (also below).

array processor—a special computer that acts as an assistant to another computer system by performing mathematical operations on arrays of numbers. Array processors are widely used in some scientific applications because they are much more efficient at performing repetitive calculations than are general-purpose computers. UNIX offers no native support for array processors, but these machines have been integrated successfully into many UNIX systems.

attached processor—any external but tightly-linked processor designed to speed the processing of specialized types of data. Examples include *array processors*, communications processors, video processors, database engines,

and the like. Although UNIX does not include any explicit support for attached processors, the system's general support for multitasking often eases the job of including such units.

backend processor—a specialized processing unit added to a computer system to speed the handling of certain types of data. It is called a "backend" unit because the ordinary user doesn't see or interact with it; all instructions pass through the main computer system. Array processors and database machines are the backend processors most commonly encountered in the UNIX community.

bang-bang controller—a controller with only one level of response, which is used when a monitored condition exceeds set limits. An example would be a furnace that turns on fully only when the temperature drops below 60 degrees and that turns off completely only whenever the mercury rises over 72. Bang-bang controllers are simpler to design and build than proportional controllers—which can order selected levels of response—but they often are less efficient.

bounded interrupt latency—a quality possessed by systems that respond to interrupts within a specified maximum amount of time. Bounded interrupt support is necessary for many types of real-time process control. UNIX, however, normally cannot guarantee such a limit, so most systems responsible for process control rely on modified UNIX kernels or resort to specially engineered precautions. See also *deterministic interrupt latency*.

computational element—one of the smaller processors in a multiprocessor system. Computational elements are especially common in systems using large numbers of processors. Also called a "processing element".

contiguous disk storage—the assignment of logically-sequential disk sectors to physically adjacent areas in preference to using an interleave factor to place them in intervening areas. Con-

tiguous disk arrangements make for faster data storage, which is often necessary for real-time systems. However, they also require the use of a high-performance disk controller, and they may require that the system process data fast or defer processing altogether because of the limited time available between the reads or writes of successive sectors.

critical resource—in multitasking computer systems, "critical resource" refers to a device, area of memory, or section of code that cannot be shared with other tasks. For example, two different tasks cannot be allowed to write to the same area of the terminal, since messages from one might overwrite or obscure messages from the other. Similarly, two tasks cannot be allowed to perform an update on a transaction file at the same time, since the result of one operation might not be interpreted properly by the other. The management of critical resources is one of the most important jobs for a multitasking operating system. If done inefficiently, total system speed may slow to little more than the rate at which jobs could be done sequentially.

data reduction—the extraction of information from raw data by the application of arithmetic and statistical transformations, or by the use of other mathematical methods. One of the goals of many real-time systems is to do any necessary data reduction at rates sufficient to allow processing to keep up with the arrival of data—by using a small amount of buffering if necessary. UNIX systems are often used for data reduction, especially by micro or minicomputers that feed mainframes with "cooked data" ready for further analysis.

deterministic interrupt latency—a defined—or computable—interval between the time an interrupt signal arrives and the time it is serviced. Deterministic latencies, or at least *bound latencies* (where the maximum delay is known), are necessary for most types of real-time control systems. Deterministic interrupt latencies normally are provided only on systems designed for process control or very simple use.

disk striping—a technique for increasing disk system transfer rates by using several disk drives in parallel to record and play back some of the bits in a multibit data word. By using eight disks, for example, transfer speeds can be increased by a factor of eight over the rates offered by the single-disk solution. Disk striping also raises the effective maximum volume size, since data is evenly spread between multiple disks.

FLOPS—an acronym for "floating point operations per second", more commonly measured nowadays as "megaflops" or "gigaflops". Because floating point calculations are more time consuming than ordinary integer arithmetic, and since they are so commonly used in scientific and control applications, a megaflops rating is often more significant than a MIPS (million instruction per second) measurement.

GPIB—initials that stand for "General Purpose Interface Bus", a connection often used in reference to the linking of electronic instruments for scientific testing and industrial work. The GPIB bus, which provides for "talkers", "listeners", and "controllers" on an 8-bit wide parallel bus, is most frequently driven by a simple dedicated system, but some UNIX-based minis and micros have also been called into service. GPIB is the IEEE 488 standard, also known as HP-IB.

host—in a computer system made up of a main processor and one or more attached processors (such as *array processors*), "host" refers to the main processing unit. In some configurations, the host might be a UNIX-driven system with the other processors treated as peripherals.

interrupt-driven—said of systems or software that respond to signals marking events. The alternatives would be to execute routines at fixed intervals or detect inputs by regularly monitoring input lines. The UNIX kernel is interrupt-driven.

interrupt handler—a routine called in response to a signal

UNIX*
JOBS
REGISTRY

National registry of candidates and jobs in the Unix field. Please give us a call; send a resume; or request a free Resume Workbook & Career Planner. We are a professional employment firm managed by graduate engineers.

800-231-5920
P. O. Box 19949, Dept. UR
Houston, TX 77224
713-496-6100

Scientific Placement, Inc.

*Unix is a trademark of Bell Labs

Circle No. 284 on Inquiry Card

marking an internal or external event requiring attention. UNIX has many interrupt handlers built into the kernel, but others often are added to respond to specific hardware configurations. Efficient interrupt handling and the assurance that delays will not exceed a maximum time are particularly important in process control work.

low-level parallelism—said of computer architectures that keep multiple copies of certain resources inside the CPU, in contrast to the normal parallel strategy of providing complete multiple processors. The low-level parallelism approach is becoming popular for RISC ("reduced instruction set computers").

MIPS—short for "million instructions per second", a measurement for specifying the speed at which a computer or system can execute programs. Naturally, since some instructions execute faster than others, the particular benchmark used to arrive at this figure is significant.

monte carlo analysis—a technique for assessing the performance or operation of systems. Based on statistical measures of a large number of events that each use a random starting value or position, monte carlo analysis is widely used in simulation work.

number crunching—the informal name for the processing of large amounts of numerical data, especially data in floating point or other complex forms. Computers designed for number crunching are likely to be different in architecture, software, and peripherals from those designed for software development or commercial use. UNIX, of course, traditionally has been used for software development.

optimizing compiler — a program that translates a high-level language into machine language, and—as part of that translation—takes steps to increase the efficiency of the code. Typically, this involves making one or more extra passes during the translation to determine how registers and other scarce system resources might best be used.

overrun—a loss of data that occurs when a receiving system accepts a new signal before it has finished receiving an earlier one. UNIX systems used in data-collection applications are prone to occasional overruns because their interrupt processing can take varying amounts of time. The buffering of inputs helps, but good design generally calls for some kind of error checking or data recording if every input is expected to be crucial.

parallelization — the decomposition of a computational task into portions that can be handled simultaneously by different processors. Unlike vectorization, parallelization does not imply that all apportioned tasks will be identical.

physical memory—the amount of memory actually present in directly-accessible form. See *virtual memory*.

poll—a check for a message or any other call for attention. Some simple microcomputer systems still poll their input ports to check for inputs, but most systems now are interrupt-driven instead.

real time—a system designed to respond to outside events, and to service them as need be. Depending on the needs of the equipment or processes being controlled, this may require response speeds as fast as a few microseconds.

An older meaning of real time was used to describe those computer systems fast enough to be interactive, in order to distinguish such systems from those that performed batch processing.

RISC—an acronym for "reduced instruction set computer", a type of processor architecture that attempts to achieve faster processing by implementing arrays of processors each with only simple instructions. RISC machines are becoming increasingly popular for scientific and engineering applications.

vectorization—the division of a computational task into many identical subtasks that can be performed by an array processor or some other processing system capable of handling multiple similar operations at the same time. Not all problems are suitable for vectorization, but—with the proper hardware—those that are can be run many times faster.

virtual memory—a feature that simulates large amounts of physical memory by swapping data back and forth between main memory and a disk storage system. One of the principal aims behind the development of the Berkeley distribution of UNIX was to add support for virtual memory to the existing version of Research UNIX. Many scientific applications—particularly those involving graphics, image processing, and data reduction—require the large memory address space that virtual memory provides.

If you have comments, questions, or corrections to offer, please send them to Rosenthal's UNIX Glossary, Box 9291, Berkeley, CA 94709.

Steve Rosenthal is a lexicographer and writer living in Berkeley. His columns appear regularly in six microcomputer magazines. ■

1986 Winter USENIX Technical Conference
Marriott Hotel — City Center, Denver, Colorado
January 15-17, 1986

TUTORIALS

For each topic area, there will be related tutorials on adjacent days, concurrent with the other technical sessions. Tutorials of general interest will also be held. Possible topics include:

- Ada Programming Language & Environment
- Window System Implementation
- SNA Networking & UNIX*
- UNIX System Internals
- UNIX Interprocess Communication, and others

Tutorial speakers will be highly qualified technical experts who are able to give an indepth presentation.

THE SPONSOR

For the latest in UNIX applications and research, people look to USENIX, a not-for-profit association of individuals and institutions dedicated to fostering the development and communication of UNIX and UNIX-like systems and the C programming language. USENIX sponsors technical conferences, produces and distributes a newsletter and serves as coordinator of a software exchange for appropriately licensed members.

THE TECHNICAL SESSIONS

The 1986 Winter USENIX Conference will consist of workshop-oriented technical sessions in three topic areas:

WINDOW ENVIRONMENTS AND UNIX
WEDNESDAY, JANUARY 15, 1986
A thorough exploration of the design and integration of UNIX-based window systems and their applications.

UNIX ON BIG IRON
THURSDAY, JANUARY 16, 1986
An analysis of issues raised by the implementation and operation of UNIX on very large, powerful mainframes, including those with multiple processors.

ADA AND THE UNIX SYSTEM
FRIDAY, JANUARY 17, 1986
An examination of the Ada language and its relationship to the UNIX system.

For complete conference information, call:
(213) 592-3243 or (213) 592-1381

Or write:
USENIX Conference Office
P.O. Box 385
Sunset Beach, CA 90742

*UNIX is a trademark of AT&T Bell Laboratories.

Circle No. 234 on Inquiry Card

RECENT RELEASES

MATURITY COMES WITH AGE

Western cultures exalt youth and often regard advancing age with horror. Eastern cultures, though, are known for placing value in age, considering that with experience comes maturity and wisdom. The computer industry usually takes a Western perspective when it comes to software, thinking that oldy software must mean moldy software. Perhaps UNIX teaches us to take a lesson from the East—the system is old, but it has been maintained over the years to serve current computer users' needs. Another piece of software, Macsyma ("macksimma"), is making the same claim.

Since it has been under continuous development at the Massachusetts Institute of Technology from the late '60s to 1982 and from 1982 to the present at Symbolics, Inc., Macsyma represents about 150 programmer-years of software design.

Symbolics now has released an enhanced version of Macsyma. As a computer algebra system used to assist scientists and engineers in solving complex mathematical problems, it boasts more than 450 licensees worldwide, making it the most widely used system of its kind. It is capable of solving integration, differentiation, Taylor series, matrix manipulations, tensor analysis, differential equations, and other mathematical problems requiring advanced calculus capabilities. The new release incorporates code developed by Macsyma users working in product design and systems analysis applications areas as diverse as acoustics, VLSI circuit design, and econometrics.

Macsyma and UNIX, however, have more in common than just their age and diversity of use. Use of Macsyma contributed greatly to Lisp being ported and ultimately integrated into 4.2BSD. Though initially available only on the non-UNIX-running Symbolics 3600 family of computers, as of this month the new version of Macsyma can be licensed for use on Sun-2 micros and will soon be available on other UNIX-based workstations. The package is also used on minis, including DEC VAXen.

Macsyma is marketed exclusively by Symbolics. For a workstation, a commercial license is $7500, and a license for government/non-profit organizations is $5250. For a VAX, a commercial license costs $15,000, a government/non-profit license, $6000.

On a VAX, Macsyma requires 1.75 MB of memory for the first user and .75 MB for each simultaneous user. About 10 MB of disk storage are required to store Macsyma's executable libraries.

Symbolics, Inc., 11 Cambridge Center, Cambridge, MA 02142. 617/577-7350.

Circle No. 282 on Inquiry Card

RTI AT MIT, TOO

An ongoing endeavor at MIT of interest to UNIX users is Project Athena—a major, campus-wide, UNIX-based program designed to integrate modern computer and communications capabilities into all phases of the educational process. Athena's principal goal is to help students learn more creatively in a wide range of disciplines, and to help improve and refine MIT's teaching methods.

Participating in this project is Relational Technology, Inc. (RTI), which announced that its Ingres relational database management and application development system has been selected as the foundation data management product for Project Athena. RTI's principal product, Ingres is sold complete with a collection of visual programming tools for accessing and displaying data and for building online interactive applications. These tools are designed to allow users to prototype and develop multiuser, shared database applications without conventional programming. Multiple levels of report writing and graphics are available. RTI's networking tool, Ingres/NET, gives users corporate-wide access to remote databases located on any computer in a network.

Relational Technology, Inc., 1080 Marina Village Parkway, Alameda, CA 94501. 415/769-1400.

Circle No. 280 on Inquiry Card

HARRIS HIGH ON HCX

Harris Corp. has announced a UNIX-based line of superminis that make use of 32-bit architecture. Entitled HCX ("Harris Com-

puter for UNIX"), the series makes its debut with the model HCX-7, which according to the Whetstone benchmark achieves 7.1 MIPS performance.

The HCX-7 operates under System V with Berkeley enhancements, and features reduced instruction set computer (RISC) innovations (160 total instructions) that typically require one machine cycle of 100 nanoseconds to execute. It comes with a five-board CPU with Schottky bit-slice technology, 32-bit addressing, and three-stage instruction pipelining. This model supports up to 32 MB of memory.

The base configuration price for the HCX-7 is $225,000, which buys 2 MB of main memory, a battery backup unit, a communications interface for 27 terminals, and a 32-user UNIX license. Adding 2 MB more of memory, a floating point processor, and an I/O expansion cabinet raises the price to $275,000.

Harris Corp., Computer Systems Division, 2101 W. Cypress Creek Rd., Ft. Lauderdale, FL 33309-1892. 305/973-5125.

Circle No. 278 on Inquiry Card

AH, SO, UNIX

AT&T and UniSoft Systems of Berkeley, CA, have entered into an agreement whereby the companies will jointly develop an interface between Kanji applications and System V running on AT&T's 3B series of computers.

AT&T UNIX Pacific Co., Ltd. (AT&T's UNIX software subsidiary in Japan), UniSoft, Nippon UniSoft (UniSoft's Japanese representative), and Argo 21 (a Japanese software house helping in the effort) will be working in Tokyo on a 3B2/400 to develop the interface, which should be completed by early 1986, and AT&T plans to license the software in source code worldwide

DaTapaSS™

Break through the PC/UNIX* barrier!

At last. A product that guarantees clear passage from IBM PC's and PC-Compatibles to UNIX — without unnecessary roadblocks.

DaTapaSS is quick and simple. No clutter, no confusion: Just a direct path to the features you need most. Features like terminal emulation. Full access to DOS functions. Error-free uploading and downloading across public networks, LAN's, or phone lines. Automatic restart and recovery of interrupted transfers. Key-selectable signon sequences. Softkeys you can program from DOS or UNIX to automate repeat transactions. It even lets you talk to other PC's . . . and Honeywell mainframes, too.

Interested? Give us a call for more information about DaTapaSS: your best route to error-free PC-to-UNIX communications.

DTSS

DTSS Incorporated
Buck Road • Box 70 • Hanover, NH 03755 • 603-643-6600
A Subsidiary of Metropolitan Life Insurance Company

* UNIX is a trademark of AT&T Bell Laboratories. IBM is a trademark of International Business Machines Corporation.

Circle No. 272 on Inquiry Card

FORTRAN 77

COMPILER INCLUDES FULL SUPPORT FOR MOTOROLA'S

MC68020/68881

- Full ANSI 77 implementation
- Full Screen Source Level Symbolic Debugger
- Unix and C Interface (Unix is a trademark of AT & T)
- Generates 68000 and 68010 Code
- Support for NS32081 and SKY FFP Math Hardware

ALSO AVAILABLE 68020/68881 MACRO ASSEMBLER

- 100% Motorola Compatible - Includes C Interface
- 2X to 20X Faster Than Most Assemblers

absoft

SCIENTIFIC/ENGINEERING SOFTWARE 4268 N. Woodward
Royal Oak, Michigan 48072
(313) 549-7111 • TX 235608

Circle No. 279 on Inquiry Card

MULTIUSER SYSTEMS?

USE THE MULTIPORT™ SOLUTION

- Connect up to 8 terminals to an IBM PC for under $100 per connection
- Network PC's together using inexpensive serial ports instead of high-priced cards
- Kits compatible with XENIX, PICK, BOS, THEOS, VENIX/86, PC-SHARE, & EasyLAN

FREE Technical Assistance!
Call (615) 254-0646

ARNET
The Multiuser Experts
476 Woodycrest Ave./Nashville TN 37210/TELEX 332762

Circle No. 271 on Inquiry Card

ASSEMBLERS

We will support you on over 20 UNIX and Xenix based machines. Targets:

Fairchild	F8/3870
Hitachi	6301, 6305, HD64180
Intel	8041, 8048, 8051, 8080 8086 family
Motorola	6800, 6801, 6805, 6809 68HC11, 68000 family
NSC	800
RCA	1802
Rockwell	6502/65c02
Texas Inst.	TMS7000
Zilog	Z8, Z80

UNIWARE™

SOFTWARE DEVELOPMENT SYSTEMS, INC.
3110 Woodcreek Drive
Downers Grove, IL 60515 U.S.A. (312) 971-8170
England: Unit C, Ltd. (0903) 205233

UNIX is a Trademark of AT&T Bell Labs
Xenix is a Trademark of Microsoft

Circle No. 273 on Inquiry Card

and offer binary versions on the 3B line through Japanese distributors. AT&T UNIX Pacific will coordinate the project while UniSoft will develop most of the software.

The project's purpose is not to accommodate translation per se, but to allow Japanese end users to make use of UNIX. Japanese users will be able to interface with System V in English using Roman characters or in Japanese using Kata-Kana characters, a phonetic alphabet commonly used in Japan. To support Japanese language input, the interface will allow a user to access an electronic dictionary of several thousand graphics-oriented Kanji characters. After the user has typed in text in Roman or Kata-Kana characters using a conventional keyboard, UNIX software will check the online dictionary and return with one or more appropriate Kanji characters. The user then will be able to select the appropriate character depending on the context of the sentence.

Save Time and Money on Data Entry

Use *ZIPLIST* to automatically look up city, state and county information based on zip code. Table of 48,000 zips allows significant savings on data entry, error corrections and file maintenance. This set of floppy disks, including easy instructions, is just $149. Most popular 5¼" and 8" formats are available. Hard disk recommended. Call or write for free information.

DCC Data Service
1990 M Street, N.W. Suite 610
Washington, D.C. 20036

Call toll-free 1-800-431-2577

In DC & AK 202-452-1419

Circle No. 274 on Inquiry Card

UniSoft already has a Kanji interface on its UniPlus version of UNIX, according to Robert Ackerman, UniSoft senior vice president. "This project is of special note because the technology involved in bringing a Kanji interface of UNIX to Japan can be translated [so to speak] to other such graphics-oriented alphabets," he said.

UniSoft Systems, 739 Allston Way, Berkeley, CA 94710. 415/644-1230.

Circle No. 277 on Inquiry Card

MUMPS IS SPREADING

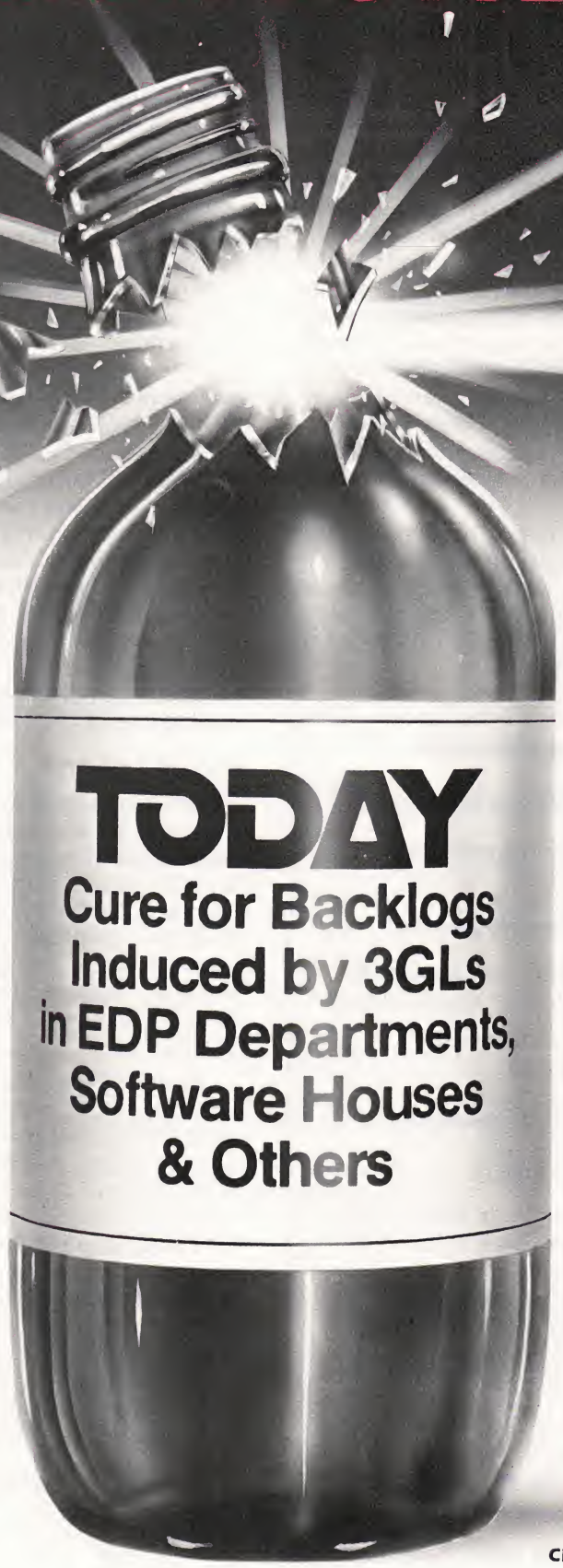
Massachusetts General Hospital is famous for many contribution to the medical world, but its contribution to the computer programming world is gaining popularity, too. MUMPS, that is, *M*assachusetts General Hospital *U*tility *M*ulti*P*rogramming *S*ystems, is an ANSI standard language developed at the prestigious institution under government contract. Motorola has announced that it is offering the Micronetics Standard MUMPS (MSM) with its Series 6000 and 2000 UNIX-based systems. These two series consist of 32-bit machines based on the Motorola 68010 processor and running Motorola's version of System V with Berkeley enhancements.

MUMPS offers software development and debugging tools including good trace capabilities and a program editor; it also executes like an interpretative language. Now available, the MUMPS licensing fee starts at $1995.

Motorola Four-Phase Systems, 10700 N. DeAnza Blvd., Cupertino, CA 95014. 408/255-0900.

Circle No. 276 on Inquiry Card

BREAK THROUGH THE SOFTWARE BOTTLENECK



UNIX™ APPLICATION DEVELOPMENT

TODAY is far more than the awkward collection of tricks and tools that are often labelled "4GL". TODAY provides a **COMPLETE application development environment** that will revolutionize the way you develop and maintain applications. **No UNIX* systems knowledge is necessary.**

Let's put it frankly: developing an application is a costly proposition. You'll need a highly skilled team of designers, analysts and programmers, and several man-years to get things off the ground. And that's not to mention the on-going costs of documentation, customization and maintenance!

TODAY tackles these problems through a new methodology with high performance architecture and a comprehensive range of features. It's so quick and easy to use that TODAY developers can do the whole job—design, analysis, development and documentation.

TODAY provides a comprehensive range of features that keep application building easy while optimizing development resources:

- **Powerful recursive logic and Decision Tables**
- **Synonyms, Menus, Prompts, Helps and Defaults for streamlined definitions**
- **Screen Painter**
- **A Report Generator which includes a Painter**
- **Push-button Self-documentation**
- **Audit Trails**
- **Source-code security through run-time only configurations**
- **Developed Applications instantly portable across UNIX* systems**

Because definitions are **Dictionary-based**, any changes are easily made in one central location. A key feature, **"tailoring"** lets you alter an application — perhaps to customize it for a particular site or user — without affecting the original version. If required, applications can be set up as Models (Prototypes) and later enhanced to grow and change with the business. Tailoring versions is the perfect solution for quickly generating multiple applications based on one Model.

TODAY runs under UNIX* or UNIX*-compatible operating systems on **super-mini down to micro business computers** using any of a range of databases. And if that's not enough, TODAY is backed by 14 man-years of research and development and the confidence of users who are breaking time zones in software development.

See us at Comdex, Nov. 20-24, Booth #434

bbj Computer Services, Inc.
2946 Scott Blvd.
Santa Clara, CA 95054
Telephone: (408) 727-4464

Circle No. 259 on Inquiry Card

*Unix is a trademark of A T & T Bell Laboratories Inc. TODAY Copyright © bbj Computer Services Pty Ltd Melbourne, Australia, July 1983

THE S SYSTEM

Continued from page 59

translate users' requests from the terminology of specific applications into the S expressions that generate the results.

Less frequently, more ambitious projects have included user-written S functions or interfaces between S and other large application systems. The relative simplicity of writing S macros means that a new system tailored to a particular user community can be written with a fraction of the programming effort required for a corresponding project using a general programming language. Also, the system development effort grows naturally (often unintentionally at first) out of direct use of S to solve user problems; there is no large initial investment in programming before any of the proposed uses can be tested and evaluated.

One of the more difficult tasks in user training has been convincing Fortran programmers that it is ordinarily not necessary to write explicit loops to operate on collections of data. Most non-programmers, however, seem to have little difficulty with the implicit iteration provided by S.

New from **Image Network**!

Documenter's Workbench®

for laserprinters and typesetters.

DWB is *troff, eqn, tbl,* and *pic* interfaced to raster printing devices.

Our existing **XROFF** product allows **DWB** to work with the following systems and printers:

- *System V*
- *Berkeley 4.2*
- *VAX/Ultrix*
- *IBM/PC MS/DOS*
- *Eunice*
- *UniPlus+*
- *System III*
- *V 7*
- *VAX/VMS*
- *Amdahl/UTS*
- *Xenix*
- *UNOS*
- *DEC LN01s, LN03*
- *APS–5 typesetter*
- *Compugraphic 8400*
- *Xerox 2700, 3700*
- *Xerox 8700, 9700*

Use **DWB** with a laser printer to make high quality documents or to make proof copies before typesetting.

Call or write to tell us *your* printing requirements!

Image Network, (415) 967-0542
448 Middlefield Road, Mountain View, CA 94043

*Documenter's Workbench is a trademark of AT&T Bell Laboratories.

Circle No. 285 on Inquiry Card

Specific user suggestions and our general recognition of the pattern of use have contributed many of the enhancements in S. An early user suggestion was the inclusion of a right-facing assignment arrow (->) for the occasion when one decides to save a result *after* typing a long expression. Our use of tools like syntax-driven parsing makes such changes easy. Interestingly, the interactive use of S in this case has direct implications for the syntax. Other enhancements that respond to user needs include: a simple mechanism to edit and rerun expressions after errors; a "diary" feature to provide a history of the expressions executed during a session; tools to help users create online documentation for their macros, datasets, and new S functions; and facilities for moving large collections of S datasets among different machines in a portable way. We have also provided a mechanism for running S noninteractively for large or repetitive analyses, and a technique for creating device-independent graphics metafiles that can be plotted later on interactive devices or batch devices. The ability to provide such facilities with only a limited expenditure of our own time derives from our modular, tool-oriented design, *and* from the similar orientation of the UNIX environment.

FUTURE PLANS

Future plans for S concentrate on improving the human interface, particularly for use with the new generation of workstations. We have made quite a bit of progress in utilizing the windows, mouse, and processor of the Teletype 5620 Terminal to produce dynamic graphics displays. These features also allow design of non-programming interfaces to data analysis, with greater flexibility and more sophisticated user support than was previously possible.

Richard Becker is a Member of Technical Staff in the Statistics and Data Analysis Research Department at AT&T Bell Laboratories. His research interests include system design, graphics, data analysis, and workstations. He and John Chambers are coauthors of S: An Interactive Environment for Data Analysis and Graphics and Extending the S System.

John Chambers is Head of the Statistics and Data Analysis Research Department at AT&T Bell Laboratories. His research interests include system design, graphics, numerical analysis, symbolic computation, and expert software. He is the author of Computational Methods for Data Analysis and coauthor of Graphical Methods for Data Analysis. ■

Copyright 1984, Association for Computing Machinery, Inc., by permission.

REAL-TIME UNIX

Continued from page 67

AST to real-time processes. The work on the scheduler and the memory-locking primitives permits response to be sufficiently "instantaneous" and predictable for a wide class of real-time applications. In UNOS, the use of *eventcounts* alone solves the same problems.

THE UNWRITTEN REQUIREMENT

Our discussion of the seven requirements would be hollow, however, if we didn't consider the practical requirements of the marketplace. Most current real-time programs are written in either assembly language or some variant of Fortran, so the market demands that these programs be kept working for as long as possible.

In the case of assembly language programs, little needs to be done. To a large extent, C already is doing the job once handled by assembler. And since C is available on almost all commercial architectures, a *complete* transition seems assured.

C programmers, though, are notorious for their disdain for Fortran. Because UNIX is written in C and because many Fortran programmers quickly embraced the new language, Fortran was often ignored in early UNIX implementations. Despite this disdain, the real-time market requires that a "production quality" Fortran accompany any real-time operating system. Like it or not, UNIX developers have come to acknowledge that Fortran continues to be the primary language used today for the analysis of data. Many installations have a large investment in existing Fortran programs. Unsurprisingly, they are unwilling to rewrite these programs simply to take advantage of the superior support UNIX offers for C.

Companies that serve this market have taken notice [4]. We believe that as the real-time programming community comes to embrace UNIX, it will move away from Fortran and switch to more modern languages such as C. But we also believe that implementations of Fortran will continue to be important for real-time work, and that they will continue to improve for some time to come.

UNIX IN REAL TIME: ADVANTAGES AND TRADEOFFS

As UNIX has migrated into the *Real-Time World*, it has changed in unmistakable ways. What has been gained and what has been lost?

What have we gained? All real-time programs must be written by a programmer, but real-time operating systems are often difficult systems to use for writing programs. For example, Digital's RT-11 and RSX-11 are rich in real-time features, but they lack the ease of use that UNIX is built around. The UNIX paradigm of using many small, simple programs to tackle big tasks has proven particularly valuable. UNIX also is unsurpassed for its wide range of available utilities. This is directly attributable to thousands of hours of tuning and "hacking" in the universities and companies where UNIX is used. One interesting note is that one of the best source language debuggers available for *any* operating system was developed for UNIX at a university, and has since become the basis for many commercial debuggers [6].

With UNIX you "get it all". One of the biggest problems a new UNIX user has comes in sorting out the myriad of choices. In addition to the usual compilers, linkers, and text editors, the system also contains presentation graphics programs, documentation development programs, and support for a wide range of peripherals. While a new RT-11 user might spend hours figuring out the *only* way to display a particular character sequence, a new UNIX user probably could spend the same amount of time figuring out the *easiest* way.

The entry of UNIX into the real-time world has brought choices to the programmer and user that never existed before.

What have we lost? Nothing, of course, comes without cost. One of the great UNIX concepts is the idea that a program can be "re-used". In many cases, this means "ported" or moved from one UNIX system to another. Although this applies to most well-written utilities, real-time programs tend to be specific to a certain task. Since there is no single conventional way to handle the "real-time" nature of a program, any UNIX software using the real-time extensions of one manufacturer is liable to run into problems if it's "ported" to another UNIX implementation. This, of course, is why many UNIX manufacturers are part of the IEEE P1003 committee to produce a standardized UNIX interface. It's hoped that through efforts such as these, real-time extensions may come to be as well defined as the **read(2)** and **write(2)** system calls currently are.

Apart from restrictions on portability, though, another problem afflicts real-time UNIX implementations. Whenever a real-time process is running, UNIX must quit acting like a timesharing system, where all processes are equal. A real-time process will tend to "hog" the resources it needs and, as a result, slow down other work on the machine. In fact, this is *exactly* what is supposed to happen, but tell that

to the lowly user waiting for a simple job to finish.

Space is another problem in the real-time realm. Real-time applications often do not need all of the "hooks" UNIX must keep in memory. There was a day when UNIX could run on 64 KB of memory; this included the entire operating system and all of a user's programs. These old systems may have been slow, but by comparison with today's UNIX implementations—many of which can hardly fit into 1 MB of memory—they seem impressively lean. To be fair, today's UNIX has grown into a system that offers much more flexibility and functionality than any of its predecessors, but along the way it has also put on some ungainly weight.

WHAT NEXT?

Adapting UNIX for real-time is an endless task. Among the many changes that could still be made we list two: *block common data* and *lightweight processes*.

Block Common Data. As mentioned previously, Fortran tends to be the programming language of choice for many real-time programmers. One structure commonly found in real-time Fortran is *block common data*, which makes available to a set of processes a "common block" residing in memory at a specific location. This *language* structure allows a hunk of data to be shared between multiple processes much like regular shared memory. Under block common data, each process uses the same "name" for a common block. Though each process has its own address for the block, all the processes share the block by name. An interesting aspect of this structure is that shared memory remains intact after a process terminates. That is, the first process that tries to access the shared memory causes the data space to be created. Other processes may access that area, and the common area will not be removed until it is explicitly released by a program or until the machine is rebooted.

Unfortunately, block common data is a hard structure to implement under UNIX. Why? The answer lies in the format of the *a.out* file. This file contains two major regions: raw program text and static data. It also includes a special segment called *bss* that is allocated at runtime. (Veteran IBM programmers should recognize *bss* as an abbreviation for "block static storage".) The text segment contains "read-only" instructions and program text. The data segment contains data that has been initialized to some value. The *bss* segment also serves as data storage that has not been pre-initialized but is allocated at program startup and initialized to the value of zero.

Most operating systems support text segments and provide for more than one *data* segment and more than one *bss* segment. These segments commonly are known as multiple *dsects*, *csects*, or *psects* (for data section, code section, or program section). Each of these *sects* is aligned by the linker and the operating system's program loader (the **exec(2)** call of UNIX) to begin on a "suitable" boundary, often a "page" of memory (where a "page" refers to the local machine's primitive unit of data storage, usually between 64 and 4096 bytes of data). The current UNIX linker, though, does not support this type of alignment; the *a.out* format does not support multiple sections and, sigh, most UNIX kernel implementations do not support segmented program layout.

The changes required to support block common data would be far reaching, affecting compilers, debuggers, and other programming utilities—as well as implementations of the **exec(2)** system call. Nevertheless, we expect to see this type of change introduced someday. (To be fair, System V already has a new file format called COFF—for "common object file format". This new binary format does address *some* of the problems of *a.out*, but there are others that are not confronted. Unfortunately, these problems cannot be discussed here since they could fill a paper of their own.)

Lightweight Processes. Much has been said in the operating system community about how asynchronous events might be correctly handled [5]. Indeed, many "modern" programming languages like Ada now provide support for more than one "thread of control" within a single process. This is so that they might accommodate asynchronous events.

Each thread can be thought of as a subprocess or a process within a process. Since all "threads" share all resources (including address space) with their siblings, they can be scheduled more efficiently than can full UNIX processes.

Lightweight processes consist of implementations of multiple threads. Each thread shares *text*, *data*, and *bss* segments with its siblings. A common stack base exists for each process, but each thread has a different stack and uses different copies of whatever hardware is needed (such as registers and **pc**) to run a "program" on the local CPU.

Unfortunately, most proposals to add lightweight processes to UNIX have become ensnarled in arguments about the amount of support they require. The result is that these processes often end up as "heavy" as the full ones offered by UNIX.

One implementation of a UNIX

look-alike called TRIX [11] from the Real Time Systems Laboratory at MIT uses lightweight processes for everything. However, the full UNIX semantics of the **fork(2)** and **exec(2)** primitives have been lost, meaning that TRIX does not look completely like UNIX for all possible programs. Indeed, the porting of programs like the Bourne shell and C shell command interpreters can be quite tricky under TRIX.

CONCLUSION

The UNIX system, brought into the world as a timesharing operating system, nevertheless has been adapted to real-time applications. In fact, because of its timesharing heritage, UNIX is rich in its support of multiprocess work, simple communication schemes, and the synchronization of processes.

Because seconds (classic timesharing time slices) are much longer than milliseconds (classic real-time increments), some facilities have had to be overhauled in order to function better in a more classic real-time system. But it is eminently reasonable to make these changes. The result, we contend, looks, feels, and works like UNIX—and retains the ease of use, excellent programming support, and documentation preparation strengths that UNIX offers.

Further work, of course, could make UNIX better still for real-time use. However, as demonstrated by commercially available implementations of real-time UNIX systems, the day of real-time UNIX already has arrived.

Clement T. Cole is an Engineering Supervisor at MASSCOMP. Previously, he consulted nationally on UNIX-related issues, and worked for various firms, including Tektronix, Inc., and the Mellon Institute of Science at Carnegie-Mellon University. Mr. Cole holds degrees in Electrical Engineering and Mathematics from CMU, and has earned an MS in Computer Science from UC Berkeley. He currently serves as a member of the IEEE P1003 POSE (UNIX) Working Group.

John Sundman is a Senior Technical Writer at MASSCOMP. Like many technical writers, he entered the computer field by way of a back door. His formal education is in Agricultural Economics, with a concentration on African farming systems. ■

REFERENCES

[1] *MASSCOMP Unix Manual*, Volume 18, *System Calls, Libraries and Macros*, Massachusetts Computer Corp. (1983).

[2] S.T. Allworth, *Introduction to Real Time Software Design*, Springer-Verlag (1981).

[3] Jeff Goldberg, *Comments on UNOS—Private Communication*, Monarch Data Systems (Sept. 1985).

[4] C. Gridley, "Improving the Performance of Scientific Applications on a Supermicro Using A Custom Floating Point Processor and An Optimizing Compiler", *Proceedings of the Summer 1985 Usenix Conference*, MASSCOMP (June 1985).

[5] B.W. Lampson, "Hints for Computer System Design", *Operating Systems Review*, Volume 17, Number 5, pp. 33-49, Xerox Palo Alto Research Center (Oct. 1983).

[6] M. Linton, "A Programming Language Debugger", *Master's Report, UC Berkeley*, (Aug. 1981).

[7] J. Ready, *Comments on VRTX—Private Communication*, Hunter and Ready (Sept. 1985).

[8] D.P. Reed and R.K. Kanodia, "Synchronization with Eventcounts and Sequencers", *Communications of the ACM*, Volume 22, Number 2, pp. 115-123, MIT-LCS (Feb. 1979).

[9] D.M. Ritchie and K. Thompson, "The Unix Time-Sharing System", *Communications of the ACM*, Volume 17, pp. 365-375, The Bell Telephone Laboratories (1974).

[10] D.M. Ritchie, "The Unix I/O System", *Unix Programmer's Manual*, The Bell Telephone Laboratories, (Nov. 1978).

[11] J. Sieber and D. Clark, *TRIX Implementation Outline*, Real Time System's Laboratory, MIT (Oct. 1982).

[12] H. Spencer, "On 4.2BSD", *Netnews—UNIX Wizards*, University of Toronto (Summer 1984).

[13] W.E. Suydam, *Off the Shelf Software Tackles Real Time Tasks*, *Computer Design* (July 1, 1985).

adax

X.25 FOR UNIX* Communications System

- Efficient, error-free data transmission to multiple hosts via international standard X.25, the only fully certified error-free public networking system used world-wide.
- User utilities
  - Remote user login
  - Remote mail service
  - Remote file transfer
- Compatible with widest number of host computers.
- Hardware available for VME, Multibus and others.
- Previously certified on TELENET, TYMNET and UNINET networks.
- Lowest cost per node.

Adax, Inc.

737 Dwight Way
Berkeley, CA 94710
(415) 548-7047

* UNIX is a trademark of Bell Laboratories.

Circle No. 283 on Inquiry Card

CALENDAR

EVENTS

DECEMBER

December 12-13 Monterey, CA: Usenix second annual graphics workshop. Contact: Usenix Conference Office, PO Box 385, Sunset Beach, CA 90742. 213/592-3243.

JANUARY 1986

January 15-17 Denver: Winter '86 Usenix Technical Conference. Contact: Usenix Conference Office (see above).

FEBRUARY

February 4-7 Anaheim, CA: UniForum International Conference of UNIX users, sponsored by /usr/group. Contact: UniForum 1986, 2400 E. Devon Ave., Suite 205, Des Plaines, IL 60018. 312/299-3131.

TRAINING

Note: Below are listed the dates, locations, titles, and contacts for UNIX-related training courses. For registration and further information on particular courses, contact the firm cited. Training firm addresses and phone numbers are listed alphabetically at the end of the calendar.

NOVEMBER

November 4-5 Edison, NJ: "Shell Programming". Contact AUXCO.
November 4-5 Santa Monica, CA: "Advanced Commands for Programmers". Contact Interactive.
November 4-6 Cherry Hill, NJ: "SNA Architecture and Implementation". Contact CSI.
November 4-8 Trumbull, CT: "Advanced C". Contact Bunker Ramo.
November 4-8 Cincinnati: "C Shell Programming". Contact ITDC.
November 4-8 Washington, DC: "C Language Workshop". Contact Structured Methods.
November 4-8 Chicago: "UNIX System Administration". Contact Uniq.
November 4-8 Washington, DC: "The UNIX System for the DP Professional". Contact Webco.
November 5-8 Washington, DC: "Programming in C". Contact ICS.
November 5-8 Los Angeles: "UNIX: A Comprehensive Introduction". Contact ICS.
November 6-8 Edison, NJ: "UNIX Tools". Contact AUXCO.
November 6-8 Santa Monica, CA: "UNIX Architecture—A Conceptual Overview". Contact Interactive.
November 6-8 New York: "Fundamentals of the UNIX System for Management". Contact LUCID.
November 6-8 Bellevue, WA: "Hands-on UNIX for Programmers". Contact SSC.
November 11-12 Tarrytown, NY: "C Data Concepts for Manager". Contact Sessions and Gimpel.
November 11-13 London: "UNIX Administration". Contact CTG.
November 11-13 New York: "Fundamentals of the UNIX System for Management". Contact LUCID.
November 11-15 Edison, NJ: "C-UNIX Interface". Contact AUXCO.
November 11-15 Trumbull, CT: "Intro to UNIX". Contact Bunker Ramo.
November 11-15 Dallas and San Francisco: "C Language Programming". Contact CTG.
November 11-15 New York and Washington, DC: "UNIX Internals". Contact CTG.
November 11-15 Santa Monica, CA: "The C Programming Language". Contact Interactive.
November 11-15 New York: "UNIX System Workshop". Contact Structured Methods.
November 11-15 Chicago: "C Language". Contact Uniq.
November 11-22 Cincinnati: "UNIX for Application Developers". Contact ITDC.
November 12 Washington, DC: "UNIX **vi** editing". Contact Webco.
November 12-15 Los Angeles: "Programming in C". Contact ICS.
November 13-14 Washington, DC: "Advanced Editing". Contact Webco.
November 13-15 Austin, TX: "SNA Architecture and Implementation". Contact CSI.
November 13-15 Tarrytown, NY: "C Data Concepts for Programmers". Contact Sessions and Gimpel.
November 15 Washington, DC: "Introduction to **nroff**". Contact Webco.
November 18-19 Dallas and San Francisco: "Shell Programming". Contact CTG.
November 18-19 London: "Advanced C Programming Workshop". Contact CTG.
November 18-19 Santa Monica, CA: "Advanced Topics for C Programmers". Contact Interactive.
November 18-19 Annapolis, MD: "The Concepts of Object-Oriented Programming". Contact PPI.
November 18-20 New York: "Office Automation". Contact LUCID.
November 18-21 Edison, NJ: "System Administration". Contact AUXCO.
November 18-22 Trumbull, CT: "C Programming". Contact Bunker Ramo.
November 18-22 Merrimack, NH: "C Programming Workshop". Contact Plum Hall.
November 18-22 New York: "C Language Workshop". Contact Structured Methods.
November 18-22 Chicago: "Unify Database Management". Contact Uniq.
November 18-22 Washington, DC: "C Language Program-

ming". Contact Webco.
November 19 Palo Alto, CA: SV Net Monthly Meeting. Contact SV Net.
November 19-21 New York and Washington, DC: "UNIX Administration". Contact CTG.
November 20-22 London: "Advanced C Programming Under UNIX". Contact CTG.
November 20-22 Dallas and San Francisco: "Using Advanced UNIX Commands". Contact CTG.
November 20-22 "Advanced C Programming Under UNIX". Contact Interactive.
November 25 New York: "UNIX System Literacy". Contact Structured Methods.
November 25-26 New York: "Using **Lex** and **yacc**". Contact Structured Methods.
November 25-28 New York: "UNIX System Concepts and Facilities". Contact LUCID.
November 25-29 London: "Berkeley Fundamentals and **csh** Shell". Contact CTG.

DECEMBER

December 2 New York: "Mainframe-to-Mini-to-Micro Links". Contact Interactive.
December 2-3 New York and Washington, DC: "Advanced C Programming Workshop". Contact CTG.
December 2-3 New York: "Shell Programming Workshop". Contact Structured Methods.
December 2-4 Edison, NJ: "Advanced C Language Programming". Contact AUXCO.
December 2-4 Santa Monica, CA: "UNIX Fundamentals". Contact Interactive.
December 2-5 Callaway Gardens, GA: "UNIX OS: The First Step". Contact AT&T.
December 2-6 Dallas and San Francisco: "UNIX Internals". Contact CTG.
December 2-6 Cincinnati: "UNIX for End Users". Contact ITDC.
December 2-6 Absecon, NJ: "C Programming Workshop". Contact Plum Hall.
December 2-6 Washington, DC: "C Language Programming". Contact Webco.
December 3 London: "UNIX Overview". Contact CTG.
December 3-5 San Francisco: "SNA Architecture and Implementation". Contact CSI.
December 4-6 London: "UNIX Fundamentals for Non-Programmers". Contact CTG.
December 4-6 New York and Washington, DC: "Advanced C Programming Under UNIX". Contact CTG.
December 5-6 Edison, NJ: "C Language Debugging". Contact AUXCO.
December 5-6 Santa Monica, CA: "Using the Shell". Contact Interactive.
December 9 New York: "Principles of Computer Graphics". Contact LUCID.
December 9-10 Santa Monica, CA: "System Administrator's Overview". Contact Interactive.

Please send announcements about training or events of interest to: UNIX Review Calendar, 500 Howard Street, San Francisco, CA 94105. Include the sponsor, date and location of event, address of contact, and relevant background information.

CONTACT INFORMATION

American Institute for Quality and Reliability (AIQR), 1494 Hamilton Ave., Suite 104, San Jose, CA 95125. 800/621-0854 ext.290, or in CA, 408/978-2911.

Asidor Training Institute, 2143 Morris Ave., Suite 5, Union, NJ 07083. 201/888-0241.

AT&T Information Systems, Institute for Communications and Information Management, PO Box 8, Pine Mountain, GA 31822-0008. 800/247-1212.

Auxton Computer Enterprises, Inc. (AUXCO), 2 Kilmer Rd., Edison, NJ 08817. 201/572-5075.

Bunker Ramo Information Systems, Trumbull Industrial Park, Trumbull, CT 06609. 203/386-2000.

Center for Advanced Professional Education (CAPE), 1820 E. Garry St., Suite 110, Santa Ana, CA 92705. 714/261-0240.

Computer Technology Group (CTG), 310 S. Michigan Avenue, Chicago, IL 60604. 800/323-UNIX, or in IL, 312/987-4082.

Communications Solutions, Inc. (CSI), 992 S. Saratoga-Sunnyvale Road, San Jose, CA 95129. 408/725-1568.

Information Technology Development Corp. (ITDC), 9952 Pebbleknoll Drive, Cincinnati, OH 45247. 513/741-8968.

Integrated Computer Systems (ICS), PO Box 45405, Los Angeles, CA 90045. 800/421-8166, or in CA, 800/352-8251.

Interactive Systems Corp., 2401 Colorado Avenue, 3rd floor, Santa Monica, CA 90404. 213/453-8649.

LUCID, 260 Fifth Avenue, Suite 901, New York, NY 10001. 212/807-9444.

Plum Hall, 1 Spruce Avenue, Cardiff, NJ 08232. 609/927-3770.

Productivity Products International, Inc. (PPI), 27 Glen Road, Sandy Hook, CT 06482. 203/426-1875.

Sessions & Gimpel Training Associates, 474 Washington Street, Holliston, MA 01746. 617/429-6350.

Silicon Valley Net (SV Net), PO Box 700251, San Jose, CA 95170-0251. 415/594-2821 (Grant Rostig).

Specialized Systems Consultants (SSC), PO Box 7, Northgate Station, Seattle, WA 98125-0007. 206/367-UNIX.

Structured Methods, Inc., 7 W. 18th St., New York, NY 10011. 800/221-8274.

Uni-Ops, PO Box 27097, Concord, CA 94527-0097. 415/945-0448.

Uniq Digital Technologies, 28 S. Water Street, Batavia, IL 60510. 312/879-1008.

Webco Industries, Inc., 14918 Laurel Oaks Lane, Laurel, MD 20707. 301/498-0722.

THE LAST WORD

Letters to the editor

SOFTWARE "DRIVER TRAINING"

Dear UNIX REVIEW,

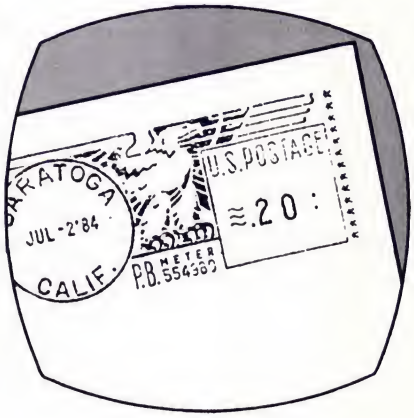
In Glenn Groenewold's *Rules of the Game* column in your July issue, he claims: "Even a completely shielded [UNIX] system doesn't approach the user ease of the modern automobile, where you need only turn the key and step on the gas." In one sense he's right; in another, he's wrong.

Mr. Groenewold has forgotten that driving is not a natural human ability. One must *learn* to drive; this normally involves a good deal of time and effort, and preferably the services of a skilled instructor. Trying to bypass this learning process is both unwise and illegal. Driving is taken so much for granted in our society that it's easy to overlook the skill and training required.

In this sense, current computer systems are already *much* easier to use than a modern automobile. Learning to use most computer systems is no harder than learning touch typing, or arithmetic, or correct English spelling. What is different is the social acceptability of lengthy training processes.

People do not want to make major efforts to learn to use a computer system, especially since every system is different. If computers become as ubiquitous as many people think, *and* there is a lot more standardization of the user interface, then eventually we may see detailed training in the use (not programming!) of computer systems become a standard part of education. Until then, all we can do is work to make inherently complex software systems simpler to use.

Henry Spencer
SP Systems
Toronto, Ontario



LEADER ZAPPED; NOW UNZAP

Dear UNIX REVIEW,

It was with mixed happiness and horror that I opened my June issue of UNIX REVIEW—happiness because my article "How it Should be Done" was prominently featured (I even got my name in Mark Compton's column!)—horror because the name of one of the most important members of the Lucasfilm Games Group was omitted entirely! *David Fox* was project leader for "Rescue on Fractalus!". The game was primarily David's idea and without his efforts there would have been no such game.

Unfortunately, the picture on page 38 of your June issue included everyone but David Fox (who was off zapping Jaggis while we were posing). To see what David Fox looks like, check the "Rescue on Fractalus!" cover picture in your local game store—that's him in the orange flight suit.

Peter Langston
Bell Communications Research
Morristown, NJ

BEG TO DIFFER

Dear UNIX REVIEW,

Joel McCormack (UNIX REVIEW, September, 1985, p. 31) comments, "There are currently no textbooks on Modula-2. . . ." I'm unsure what McCormack classifies as a "textbook". The following most certainly exist:

1) *Interactive Programming Environments*, by Barstow, Shrobe, and Sandewall (McGraw-Hill, 1984);

2) *Modula-2 for Pascal Programmers*, by Gleaves (Springer-Verlag, 1984);
3) *Programming in Modula-2*, 3rd ed., by Niklaus Wirth (Springer-Verlag, 1985);
4) *Software Engineering With Modula-2 and ADA*, by Wiener and Sincovec (Wiley, 1984);
5) *Modula-2 Programming*, by Ogilvie (McGraw-Hill, 1985).

Bill Freeman
Tinton Falls, NJ

Thank you for bringing these resources to light. In a similar vein, it has been observed that a reference within the same article incorrectly indicated that "Modula-2—A Solution to Pascal's Problems" by Sumner and Gleaves had run in the September, 1983, issue of SIGPLAN. The article in fact was printed in the September, 1982, issue of that publication. *Editor*

GOT UP AND GONE

Dear UNIX REVIEW,

I *wondered* why, in the *C Advisor* (March 1985), Bill Tuthill felt it necessary to apologize for his use of **goto**s. Sure enough, in the June letters to the editor, the **goto** witch hunters were out in force.

Just so there is no misunderstanding, I agree that the **goto**s of BASIC and Fortran are an abomination. **Goto**s in C, however, are *seldom* a problem. Those who claim that C **goto**s are just like Fortran **goto**s are chasing a straw man.

Bill Tuthill, even if he never does anything else in his career, has long since "paid his dues" as a professional programmer. His instincts are right on target.

Both letter writers would have us believe that the sample code is improved by turning it into a formal loop. That implies that the *nature* of the code fragment is *iterative*; it says that the programmer wants to *emphasize* that this is an iterative algorithm.

The fact is that the code would *normally* be executed only once. Making a loop out of it is a smokescreen that obscures the meaning.

If the anti-**goto** folks are to be consistent, they'll have to take on much more than just literal **goto**s. An obvious target is **setjmp/longjmp** (non-local **goto**s with automatic call frame abort). Exceptions also have to go; instead, all programs must have a **main** loop somewhat like the following:

```
while ( ! divide_by_zero && ! illegal_instruction && ! hangup etc.
```

Of course, if any of these exceptions *do* occur, they must be passed back, not just one *function* at a time, but one *level* within each function at a time. Interrupts and pipelining are even worse—rather like three-dimensional **goto**s, out of the plane of the program listing. Can them too. And as for *parallel processing*—that's like four-dimensional **goto**s! Anathema!

I wonder what those guys make of the new ANSI C proposed storage class **volatile**—the one that warns the compiler that some other process may modify a data item?

There *is*, in fact, a piece of very badly structured code in Bill's example, which both letter writers missed. I refer to the "exit(1)". Obviously some error has occurred. There is no diagnostic; there is no attempted cleanup. Instead, the program executes a **longjmp** with an unknown status to an unknown context, with a faint prayer (the "1") that that context might do something sensible. Good luck.

Dave Fafarman
ENSCo Engineering Software
El Sobrante, CA

Dear UNIX REVIEW,

I opened my first issue of UNIX REVIEW to discover a letter decrying the use of the **goto** statement. I will not argue that the **goto** is a crutch for inadequate programmers. But please, *please*, do not forget the "kludge" programmer in C. This level of programmer needs as many crutches as he can get to get jobs done!

I am a "kludge" programmer. I get jobs done. My programs may not be sophisticated, but they do work. If I write a program that is going to be used frequently enough to warrant it, I'll have it rewritten by a PROGRAMMER. (That's all capital letters, and no smile!) I highly respect the type of person who has the discipline to write concise and efficient code. But I am a mechanical engineer and not a PROGRAMMER. I do not apologize for this fact and I highly resent it when a tool, such as C, is closed to me by the search for "purity".

I have been writing programs in C since January. I find it to be a highly useful tool. As far as I know, C and FORTH are the only widely available languages that were written by people who needed to get jobs done...period. The other languages were either written to teach people "good" programming style or to support antiquated I/O systems. I am usually the person dragging my (antiquated) portable 8-bit machine around in order to have a FORTH system available to me so that my jobs will get done on time. This has changed considerably since January. I hope that this trend will continue.

Lew Merrick
Lynnwood, WA

ADVERTISERS' INDEX

COMING UP IN DECEMBER

International UNIX

- **The American Bias**
- **International layers**
- **The stakes**
- **Shrinking the world down to size**
- **Market update**

ZIM 2.5 A DBMS REVOLUTION

Have you been looking for perfect data management that you can enjoy on your own terms? Then you've probably already heard of ZIM 2.4 — the most powerful database system available. Until now. Because ZIM 2.5 is here.

ZIM 2.5 is a fourth generation application development tool which makes it possible to expand the capabilities of your micro beyond what you've ever imagined. ZIM mirrors the complexities of the real world by letting you develop as many and as varied applications as you could possibly need.

"ZIM is...a successful migration of mainframe ideas and needs to a micro. (ZIM) proves not only that the job can be done but also that it can be done well. ZIM provides a reference against which current and future data bases can be judged." James Creane, Data Based Advisor/July 1985.

Speed
ZIM breaks the speed limit — between 3 and 50 times faster than industry leaders in sorting and joining files within the data-base. ZIM's internal architecture, and the implementation of its strategy analyzer and priority-driven buffering ability, ensure that data is processed in the most efficient manner possible.

Portability
ZIM is the only database management system with 100% application portability for single-user and multi-user configurations. ZIM is available under PC-DOS, Concurrent PC-DOS, UNIX, XENIX, and QNX. Never again will you be required to re-write your applications for different operating systems environments.

Power
ZIM's high-level language lets you build user commands which implement applications without the necessity and cost of additional programming tools. ZIM's forms facility and extensive report generator permit completely menu-driven applications. Completed compiled, applications use the Runtime System, leading to fast execution, preventing unauthorized access or modifications, and decreasing cost and memory requirements.

Flexibility
ZIM gives you unprecedented simplicity and flexibility. ZIM commands parallel simple English sentences, making it easy to learn and use. Other features include automatic updates of all indexes, multi-user support, and an extensive range of validation, editing and masking facilities. ZIM's limits are only those of your hardware, operating system and imagination. And with ZIM 2.5, your database is no longer limited to a single hard disk.

"ZIM is (a) well-conceived, soundly-implemented, thoroughly professional system. Its design evidences a strong commitment to consistency and to the goal of natural nonprocedural user interaction."

Richard M. Foard, PC Tech Journal, October 1985.

ZIM 2.5 — DATA MANAGEMENT AT ITS BEST

Circle No. 249 on Inquiry Card

1785 Woodward Drive
Ottawa, Ontario K2C 0R1
(613) 727-1397

ZANTHE
INFORMATION INC.

Available now from IBM as Interactive Executive (IX) Basic on the IBM Interactive Executive for System/370 (IX/370)

The Language for a New Generation

Portability. UX-Basic™ application programs execute unchanged on any UNIX™ machine and are completely device independent.

Power. UX-Basic contains the building blocks for efficient application program development. It also allows you to tap the full power of UNIX and gives you direct access to data bases.

Productivity. UX-Basic is friendly and easy to learn and use. The interactive programming environment provides syntax checking as well as real-time debugging and testing.

Performance. UX-Basic gives you speed when you need it with our efficient pseudo-code compiler/runtime package. We are constantly working to keep UX-Basic's performance at the leading edge.

Profit. UX-Basic programs are structured, modular and readable. Maintenance and support are easy.

Perfect for UNIX . . . a new generation of computers . . . a new generation of computer users.

UX Software, Inc.

10 St. Mary Street, Toronto, Canada M4Y 1P9
Tel: (416) 964-6909 TLX: 065-24099

Available from major computer manufacturers such as Altos, AT&T, Siemens and an international network of distributors.

UniForum.
The International Conference of UNIX Users
February 4-7, 1986

UNIX is a trademark of AT&T Laboratories.
UX-Basic is a registered Trademark of UX Software, Inc.

See us at
COMDEX/Fall '85
November 20-24, 1985
Las Vegas Convention Center-West Hall
Las Vegas, Nevada

Circle No. 235 on Inquiry Card