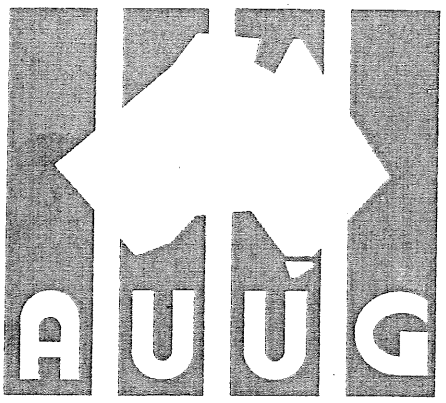
ISSN 1035-7521




# Australian UNIX systems User Group Newsletter

# AUUGN

Volume 12, Number 1

Registered by Australia Post, Publication Number NBG6524

# AUUG Inc. Newsletter

# AUUGN

## Volume 12 Number 1

## CONTENTS

AUUGN is the journal of AUUG Incorporated, an organisation with the aim of promoting knowledge and understanding of Open Systems including but not restricted to the UNIX system, networking, graphics, user interfaces and programming and development environments, and related standards.

# AUUG General Information

## Memberships and Subscriptions

Membership, Change of Address, and Subscription forms can be found at the end of this issue.

All correspondence concerning membership of the AUUG should be addressed to:-

The AUUG Membership Secretary
P.O. Box 366
Kensington, N.S.W. 2033
AUSTRALIA

Phone: (02) 361 5994
Fax: (02) 332 4066

## General Correspondence

All other correspondence for the AUUG should be addressed to:-

The AUUG Secretary
P.O. Box 366
Kensington, N.S.W. 2033
AUSTRALIA

Phone: (02) 361 5994
Fax: (02) 332 4066
Email: auug@munnari.oz.au

## AUUG Executive

President **Pat Duffy**
*pzd30@juts.ccc.amdahl.com*
Amdahl Australia Pty Ltd
New South Wales

Vice President **Chris Maltby**
*chris@softway.sw.oz.au*
Softway Pty Ltd
New South Wales

Secretary **Peter Barnes**
*pdb@uqcspe.cs.uq.oz.au*
Computer Science
University of Queensland

Treasurer **Michael Tuke**
*mjt@anl.oz.au*
ANL Limited
Victoria

Committee **Frank Crawford**
*frank@teti.qhtours.oz.au*
Q.H. Tours Pty Ltd
New South Wales

**Andrew Gollan**
*adjg@softway.sw.oz.au*
Softway Pty Ltd
New South Wales

**Peter Karr**

New South Wales

**Scott Merrilees**
*sm@bhpese.oz.au*
BHP Information Technology
New South Wales

**Stephen Prince**
*sp@labtam.labtam.oz.au*
Chancery Lane Computer Services Pty Ltd
Victoria

## Next AUUG Meeting

The AUUG'91 Conference and Exhibition will be held from the 24th to the 27th of September, 1991, at Darling Harbour, Sydney. The AGM of AUUG Inc. will be held during the conference.

The AUUG'92 Conference and Exhibition will be held from the 8th to the 11th of September, 1992, at the World Congress Centre, Melbourne.

# Editorial

This rather belated issue of AUUGN will be my last as Editor. Due to pressure of work I have been unable to devote as much time to the task of editing the newsletter as I would have liked. In addition, I am heading overseas soon, and so, regretfully, I am giving up the editorship.

The new editor shall be Jacinta Crawford. She can be reached by through the same post office boc and e-mail address listed below.

I have enjoyed my time as editor, and I hope that when I return from OS I will be able to contribute further to the running of AUUG Inc.

## AUUGN Correspondence

All correspondence regarding the AUUGN should be addressed to:-


David Purdue
AUUGN Editor
PO Box 366
Kensington, NSW, 2033
AUSTRALIA

| | |
|---|---|
| Email: | auugn@munnari.oz.au |
| Phone: | +61 3 353 3913 (w)<br>+61 3 813 1258 (h) |
| Fax: | +61 3 353 2987 |


## Contributions

This Newsletter is published approximately every two months.

Contributions should be sent to the Editor at the above address.

I prefer documents to be e-mailed to me, or mailed to me on a floppy disk (IBM-PC 5-1/4 inch or 720K 3-1/2 inch; or Macintosh 3-1/2 inch), and in plain text format. Hardcopy submissions should be on A4 with 30 mm left at the top and bottom so that the AUUGN footers can be pasted on to the page. Small page numbers printed in the footer area would help.

## Advertising

Advertisements for the AUUG are welcome. They must be submitted on an A4 page. No partial page advertisements will be accepted. Advertising rates are $300 for the first A4 page, $250 for a second page, and $750 for the back cover. There is a 20% discount for bulk ordering (i.e., when you pay for three issues or more in advance). Contact the editor for details.

## Mailing Lists

For the purchase of the AUUG mailing list, please contact the AUUG secretariat, phone (02) 361 5994, fax (02) 332 4066.

## Back Issues

Various back issues of the AUUGN are available, details are printed at the end of this issue.

## Acknowledgements


This Newsletter was produced with the kind assistance of and on equipment provided by the Advanced Imaging Systems department of Kodak (Australasia) Pty Ltd. I would also like to thank Labtam Australia for providing me with a network connection.

## Disclaimer

Opinions expressed by authors and reviewers are not necessarily those of AUUG Incorporated, its Newsletter or its editorial committee.

# AUUG Book Club Book Reviews

These books have been reviewed under the AUUG Book Club scheme. Review copies were kindly supplied by Prentice Hall Australia Pty Ltd. These books can be ordered from Prentice Hall using the order form that appears on page 11. Don't forget - a 20% discount applies to AUUG members!

## UNIX FOR VMS USERS

by Philip E. Bourne

ISBN 0-13-947433-1
368 pp, RRP $84.95, 1990

*Review by Vicki Jordan*
*LaTrobe University Computer Centre*
*<ccvj@lure.latrobe.edu.au>*

This book is not designed as an introduction to UNIX for first time computer users, it is specifically targeted at experienced VMS users who are beginning with UNIX. As such, its whole "tutorial" on UNIX is presented as a comparison between VMS commands and their UNIX equivalent. A side benefit is that it also serves to expose the experienced UNIX user to VMS!

There are 13 chapters:

Introduction:

Fundamentals: System internals, command structure and file naming, device, directory and file structure, special characters, wildcards.

Getting Started: Terminal characteristics, user environment, help, documentation, command line editing.

File Management: Common file manipulation commands.

Editing: Intro to vi, sed, ed, awk.

Communication: mail, talk, write.

System Resources: who, ps, du, vmstat, netstat, kill, at, nice/renice.

Devices, Queues, Background Processing: /etc/printcap, lpr, lpq, tar, dd, &, jobs.

Advanced File Management: ls flags, more, C shell extensions, chmod, chgrp, cmp, diff, find, grep sort, tr, ln.

Programming: Compiling and linking, make, dbx, prof, ar.

Shell Programming:

Text Processing:

Processor-Processor Communication: rlogin, rsh, rcp, telnet, ftp, tip, uucp.

Each chapter consists of an introductory section which attempts to place the topic into general correspondence with a reference to the nearest equivalent VMS command or set of commands. Then the corresponding UNIX command is examined. The chapters end with a summary which usually consists of a detailed examination of the output of a set of commands. The entire text is interspersed with comparative tables. For example, the section on debugging presents, side by side, the output from invoking RUN on a VMS file compiled with DEBUG and the output from dbx invoked on the file compiled with f77 -g. "SET BREAK" is shown against "stop", "SHOW BREAK" against "status", "EXAMINE" against "print"...etc.

There are 4 appendices plus a glossary:

Command Summaries: VMS Commands vs UNIX Equivalents.

Editor Summaries: VMS EDT Line Mode vs UNIX ex and vi.

Important UNIX files: ...and their purpose.

Additional Reading: ..categorised into beginners, intermediate and advanced user, System Administration, UNIX for Micros, System V, BSD, plus references for individual chapters. This bibliography in itself is very useful.

Summary:

I have no hesitation in recommending this book to experienced, and perhaps even casual, VMS users who are beginning with UNIX. It is well organised, very well presented and excellently indexed. My only criticism is its concentration on BSD and C shell, which is, I suppose, to be expected since Ultrix is a Digital Equipment Corporation product.

# UNIX SYSTEM ARCHITECTURE

by Prabhat K. Andleigh

ISBN 0-13-949843-5
274 pp, RRP $57.95, 1990

*Reviewed by Pauline Khoo*
*Advanced Imaging Systems*
*Kodak (Australasia) Pty Ltd*
*<khoo@Kodak.COM>*

Andleigh's book *UNIX System Architecture* presents a fairly comprehensive description of, as its title suggests, the architecture of the UNIX operating system.

The book starts of with a general introduction to multi-user operating systems concepts in Chapter 1. This chapter is useful for readers not already familiar with these concepts, but may be skipped by more advanced readers.

Chapter 2 proceeds to introduce the reader to the UNIX kernel sub-systems. Here, the author first presents a functional overview of UNIX, which describes the capabilities of the kernel sub-systems. This is followed by an architectural treatment, which describes the layout of the modules of each sub-system and the inter-module interactions.

Chapter 3 is the main chapter of the book. It starts with a fairly detailed description of control structures and tables used by the UNIX kernel in managing processes. This is followed by an architectural treatment of the kernel sub-systems including memory management, process management, interprocess communication, file system management, input/output management and networking. The algorithms for some of the key routines of these sub-systems are also provided.

The following chapters briefly cover related UNIX issues relating to distributed and multi-processor systems, system performance, UNIX system porting, implementation differences and application portability.

Andleigh's book would be informative to readers at a variety of levels, particularly to novices in multi-user operating systems and UNIX. It provides more than a basic level of understanding of the UNIX operating system, though not as detailed as the more established book, *The Design of the UNIX Operating System* by Maurice Bach. It is well written and extremely readable.

# UNIX DATABASE MANAGEMENT SYSTEMS

by Ulka Rodgers

ISBN 0-13-945593-0
338 pp, RRP $39.95, 1990

*Reviewed by David Newton*
*Q.H.Tours*
*<dave@teti.qhtours.oz.au>*

Almost without exception, modern business computing is irrevocably linked with the need to store, process and retrieve large volumes of data. Once the domain of mainframes, commercial database applications have now penetrated all levels of computing with a continuing need for increased productivity and functionality. With the current interest in *open systems strategies* and the greater acceptance of UNIX as a commercial operating system it is surprising how little practical literature has been written on the subject of database management in the UNIX environment.

*UNIX Database Management Systems* is first and foremost a practical book on commercial DBMS applications for the UNIX world. It has the type of practical information that everybody wants to know but nobody is prepared to tell you. It is a useful book for those who work with and develop these types of systems.

The book strives to achieve two main goals:

- To explain the influence a UNIX operating system has on a DBMS application.
- To discuss the tradeoffs faced by database designers in developing these applications.

There is no precursory discussion on UNIX in this book, it assumes the reader has a prior knowledge of both database theory and UNIX. The book is suitable for database designers and administrators and those wishing to gain a background in the subject, like MIS managers.

The book is divided into five sections containing fifteen chapters in total, three appendices and a bibliography.

**Part 1** is an introduction to the book and sets out to lay a theoretical foundation for the rest of the text. Its aim is to provide a technical background for later discussions on practical issues. The section begins with a historical discussion of databases and leads into a description of various theoretical DBMS models. Because of the profusion of UNIX relational DBMS's a chapter is given to relational concepts which includes the always present discussion on data normalization. The section concludes with the benefits of a DBMS and the final chapter sings the praises of the **SQL query language.**

**Part 2** deals specifically with the relationship of a DBMS to its UNIX environment and how the use and

abuse of that environment can effect the DBMS's design and behaviour. The section is divided into three parts.

- A brief description of UNIX facilities including such things as system processes, physical memory, disk storage, device drivers, concurrency control, networking, kernel buffers and assorted UNIX utilities. For the seasoned *UNIXer* this chapter may not reveal anything special.
- In contrast to the previous chapter the focus is on the DBMS and its facilities and design considerations. The chapter covers storage methods, access methods, data integrity systems, concurrency controls, application builders, host languages and fourth generation languages.
- The final chapter in this section joins together the previous two chapters raising assorted issues relating to the running of a DBMS under UNIX. The practical tip provided in this part of the book are numerous and are worth thinking about, even by the best of us.

**Part 3** reviews four of the most significant UNIX RDBMS on the market.

- Informix 3.3
- Ingress 5.3
- Oracle 5.1
- Accell/Unify 1.3/4.0

Each RDBM's is appraised in a similar way and begins with a discussion on the history of the company and the product itself. Issues discussed are product packaging, product modules, data control systems, application builders, utilities, special feature and the products ability to integrate with UNIX. The use of tables of product features is prominent in this section and all vendor specific terminology is translated for the readers.

This section gives an interesting comparison of the products but is already dated and is no substitute for information direct from the vendor.

**Part 4** of the book is about selecting the appropriate DBMS to meet the users needs and is contrasted with the previous section. In the words of the author; *this part of the book attempts to arm you against the temptation of succumbing to the best sales pitch.*

The three chapters of this section describe the basic selection process and cover:

- Determining application requirements
- Assessing the tradeoffs
- Benchmarking

Selecting a product can be a difficult task, particularly if you have not done it before and one of the great assets of this book is the simple rule of thumb approach used to guide the readers.

**Part 5** is the final section of the book and is entitled **Future Directions.** It attempts to leave the reader with a warm friendly outlook on the future of database systems but as the author says; *take these flights of fancy with a little grain of salt: They are based on what we know today. Tomorrow will probably be different.* Actually this section is not as fanciful as made out and outlines the future of distributed databases, expert systems, case tools and natural languages. The discussion on distributed databases is not detailed and considered to be outside the scope of the book.

**Conclusion:**

Like a well constructed database application *Unix Database Management Systems* is concise and well constructed. The plentiful tips on tuning databases are like nuggets of gold to those who are new to building and maintaining these types of systems. It is disappointing that **SYBASE** was not included in the product comparison in chapter three but as time proceeds the product information will become irrelevant anyway.

For those with an interest in commercial UNIX databases, a book well worth reading!

## AN INFORMIX 4GL TUTORIAL

by Paul Mahler

ISBN 0-13-464173-6
282 pp, RRP $43.95, 1990

*I could not find a volunteer to review this book; I may send a review myself for a future issue. - Ed.*

## MULTIMEDIA APPLICATIONS DEVELOPMENT WITH THE ANDREW TOOLKIT

by Nathaniel S. Borenstein

ISBN 0-13-036633-1
310pp, RRP $72.95, 1990

*Reviewed by Matthew Barry*
*Department of Computer Science*
*University of Melbourne*
*<meb@cs.mu.oz.au>*

Multimedia applications are becoming very fashionable in the early nineties. The idea is that information should be presented in a form, or forms, most accessible to humans. This may mean that the application uses a graph to illustrate changes in temperature, or full colour pictures from a CD ROM to show what a tiger looks like. There is an enormous, and growing, variety of computer-based media. The Andrew Toolkit is designed to provide a flexible architecture for multimedia applications development.

The author starts the book by giving a couple of examples of multimedia applications. The reader is told that Andrew makes it "relatively easy to create such multimedia applications". The section on multimedia applications ends with the warning that "it is worth remembering that the ultimate rationale for these technicalities [the content of the rest of the book] is the creation of fancy multimedia applications that fit seamlessly together". The warning is appropriate as this is just about the last time that multimedia applications are mentioned in the book. Apparently a secondary aim of the book is to teach object-oriented programming.

The book consists of five chapters and five appendices. Approximately half the pages are used in the appendices. In each chapter and in two of the appendices the author provides exercises and programming projects for the reader. Solutions for a selection of the exercises are provided in Appendix E. A few of the exercises are not related to learning how to use the Andrew Toolkit, but instead are trivial programming exercises. The book is full of example programmes. Sometimes reading the book is like reading a heavily commented code listing. Source code for the examples in the book is distributed along with the Andrew Toolkit. Unfortunately I have been unable to get the Andrew Toolkit running on our Sun 4 and so I am unable to test the examples.

The five chapters of the book introduce the conventions and basic classes used to build applications with the Andrew Toolkit. At the end of the tour the diligent reader should be able to construct programmes using the Andrew Toolkit. The author does not provide a complete summary of the conventions used in programming with the Andrew Toolkit, which is a pity as

there is a deal of convention to be followed. The primary example throughout the book is the dreaded "Hello World" programme. Indeed I have never seen a dead horse flogged harder. The poor scope of the example programme means that many Andrew Toolkit features are introduced into the book without any motivation. The author simply says lets add menus, or whatever, to the "Hello World" programme. Anyway by the end of the five chapters the usual user interface toolkit subjects (menus, windows, mouse input, etc) are covered. Along the way some pearls of wisdom about the philosophy of using the Andrew Toolkit are imparted and some arcane terminology is explained. The author provides no pointers to other information either about Andrew or multimedia applications development in general.

The book is liberally provisioned with sample code. At times a couple of pages of sample code are inserted to show that one or no lines of code are different. It is nice not to have to take an author's word for things. As the "Hello World" programme is developed the author often would be better off inserting the text of the new procedure, or modified lines, rather than forcing the reader to wade through a couple of pages of code to find the piece of code under discussion.

Appendix A is simply an example programme demonstrating how to use the Andrew Toolkit classes for getting more complex answers from the user.

Appendix B is designed to show the "elegance and power" of object-oriented programming. It is interesting to see how the author goes about constructing a more complex programme. Most of the text describes the code rather than the design processes used by the author. Things do turn out nicely using the Andrew-style of object-oriented programming. In fact the contents of Appendix B may have made a good primary example for a lot of the book.

Appendix C is a fifty page "grand tour" of the Andrew Toolkit. Basically this consists of a single sentence description of the class and a list of the methods exported from the class for some of the classes. I would find it surprising if the Andrew Toolkit does not provide some facility to interactively browse around the manual pages for its large class hierarchy.

Appendix D gives some hints on debugging applications running under the Andrew Toolkit. On the face of it this is a pretty useful chapter as having to debug dynamically loaded modules is not part of every day C debugging experience. However I again would find it surprising if the issues involved in debugging Andrew Toolkit classes are not covered by the accompanying documentation. The more general advice given is old hat, e.g. using print statements. Borenstein also mentions four commonly encountered errors and the symptoms associated with them.

The typographical conventions in the book are dreadful. The example code seems to have been formatted to one page width and then restricted to a smaller page width with the resulting line wrap being ignored. Occasionally this makes the code hard to read. There are several instances of the first line of an exercise being tucked away underneath an example programme and the rest of the exercise being over the page. There is even one instance of a heading being the last line on a page. In general I feel that very little care has been taken with the production, or proof-reading, of the book.

If you are looking for an insight into the multimedia applications development and the experiences other people have had with them, definitely this is not the book to buy. On the other hand if you want an introduction to using the Andrew Toolkit then this book may be of some use.

# AUUG BOOK CLUB & PRENTICE HALL AUSTRALIA

## 20% DISCOUNT TO AUUG MEMBERS

**Please send me copy/copies of the following books —**

| | | | | |
|---|---|---|---|---|
| ☐ | Bourne/ **Unix for VMS Users** | | | |
| | RRP $84.95 | ISBN: 1394-7433-1 | Cloth | 1990 |
| ☐ | Andleigh/ **Unix System Architecture** | | | |
| | RRP $57.95 | ISBN: 1394-9843-5 | Paper | 1990 |
| ☐ | Rodgers/ **Unix Database Management Systems** | | | |
| | RRP $39.95 | ISBN: 1395-0353-6 | Paper | 1990 |
| ☐ | Mahler/ **Informix 4GL Tutorial** | | | |
| | RRP $43.95 | ISBN: 1346-4173-6 | Paper | 1990 |
| ☐ | Borenstein/ **Multimedia and Application Development with the Andrew Toolkit** | | | |
| | RRP $72.95 | ISBN:1303-6633-1 | Paper | 1990 |

***Deduct 20% from listed retail price.**

Name: _____ Organisation: _____

Address: _____
(Street address only)

_____ Telephone: _____

☐ Please send my book/s on 30-day approval *(tick box)*

Enclosed cheque for $ _____ (Payable to 'Prentice Hall Australia')

Please charge my: ☐ Bankcard ☐ Visa ☐ MasterCard

Credit Card No: ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐

Expiry Date: _____ Signature: _____

Mail completed order form to Prentice Hall Australia, PO Box 151, Brookvale NSW 2100

OR **Use our FAST PHONE SERVICE call Liz Guthrie.**
***Have your credit card ready (9am to 5pm)***
**SYDNEY (02) 939 1333**

**Prentice Hall Australia Pty Ltd**
**7 Grosvenor Place, Brookvale NSW 2100**
**Tel: (02) 939 1333 Fax: (02) 938 6826**
A PARAMOUNT COMMUNICATIONS COMPANY

# (Ex) President's Letter

This letter was sent by Greg Rose, who resigned the Presidency of AUUG Inc shorty after the AUUG'90 conference.

It's safe now for me to write a letter for the AUUGN, since you only get to write one ex-president's letter; I can't set a precedent!

It was with some amount of sadness that I tendered my resignation at the last meeting of the executive committee. AUUG has done a lot for me over the past *ahem* years, giving me many new friends, interesting moments (such as stand-up brawls over balloons), more knowledge of the world of computers, and I could go on for hours.

I've tried to repay some of that by taking the group in directions which I felt should simultaneously:

1. Increase benefits to members;
2. Increase the group's membership;
3. Help the UNIX community at large to grow.

Properly implemented, these three goals should become self reinforcing, feeding back to enable clear growth and better services.

I don't think that these goals have been addressed as well as they could have been; which doesn't mean that anyone has failed, either. It is in the nature of things, particularly with a volunteer organisation like AUUG, that people simply run out of energy, time or bluster. With the UNIX marketplace growing steadily (some would say slowly) at 45% per annum, we are forced to the conclusion that AUUG has to change significantly, and to keep changing, to meet the requirements of its members. If only we had had Lionel Singer's crystal ball, we could have set up an entirely different AUUG.

In case you hadn't noticed, the previous paragraph was an attempt to point out that we need 45% more people willing to get in there and do things for the group. We're finally sorting out and offloading some of the drudgery, but innovations never come easily. We can (and do) look to other UNIX user groups for ways to meet our aims, but not all of their methods can be made to work here. Australia is more isolated; more centralised in two cities and yet geographically far flung; still coherent (as opposed to the U.S. where the user groups have long since stopped talking to each other); and small. We here in Australia have to address these problems in our own ways.

Now a bit of maudlin retrospective. The group started way back in about 1976, with people mostly from the University of New South Wales, although that didn't last long. These people got together to exchange programs and bug fixes, and to talk about Unix (it wasn't capitalised then). Strangely enough, the reason for talking about it usually came back to the fact that for us, as programmers, Unix was a lot of fun to use, and access to the full source code was a great way to learn. Almost every aspect of the preceding statement is now inapplicable. UNIX is still fun for programmers, but they are a decreasing proportion of our membership (I'm only tenuously a programmer these days). You generally don't get the source code to look at, the number of bugs is increasing in proportion to the size of the system (which has recently tripled), but the bug fixes are becoming rarer.

Through the years, we went through a barrage of the same, standard questions:

- What's a UNIX? (snicker)
- Will UNIX ever take over the world?
- I'm glad to see UNIX is finally doing something...
- `goto loop`

There were always people ready to throw tomatos at UNIX converts. Of course these were usually in response to the watermelons going the other way. I think that, like most new technologies, UNIX needed to evolve softer corners, while other (I almost said older, but of course this isn't really the case) systems needed to realise that UNIX had a lot to offer, hidden behind a sign saying "You are not expected to

understand this."

Gradually, Unix became Unix™, then the UNIX™ Operating System, then System V™ (at least according to the owners). Other people had other words, but it isn't the name or the version of source code which means survival for UNIX. It is the simplification and clarification which UNIX brought to bear on something which had been (and is again) considered to be a hard problem. I think it is indicative of the subtlety of UNIX that it was ten years after the paper announcing Unix in the Communations of the A.C.M. that Dennis Ritchie and Ken Thompson were given the A.C.M.'s Turing Award.

During all this time UNIX continued its steady growth, until now people are astounded that it is possible to have fifteen years experience with this "new" operating system. And obviously all operating systems have hierarchical directory structures, but why did UNIX use "/" instead of "\e"?

Enough of that. I'd like to take this opportunity to thank all of the people (and companies) who contributed to the success and growth of AUUG over the past few years. They all made my job easy. The new organisation has Pat Duffy heading up an excellent and expanded executive, with old hands and new faces combining, I think, to really develop the group. I wish them all the best, and hope you'll all support them.

What of me? I felt that I was stagnating to some extent, that my style of (non-)management might have inflicted itself on Softway and AUUG long enough. So I'm off on a research jaunt for a year to sharpen my cutting edge. AT&T, in a brilliant strategic move, have allowed me to go to IBM's Thomas J. Watson Research Center, about an hour's drive north of New York, where I'll be working on distributed services for supercomputer applications. I'll be more than happy to play host to my many friends from AUUG (one at a time) if you happen to be in the area.

Best regards,

Greg Rose.

# Beyond SCCS

"Beyond SCCS" was a talk given by Tim Roper at the AUUG Summer'91 Victoria conference held at Monash University. After a number of requests, Tim has consented to having the slides from his talk reproduced here.

91/02/18 (1.8)
Beyond SCCS
FOIL 1

## Beyond SCCS

or

Basic
Software Configuration Management

Done Cheaply with UNIX Tools

Timothy Roper

Senior Systems Programmer

Labtam Australia

*timr@labtam.oz.au*



91/02/18 (1.8)
Beyond SCCS
FOIL 2

## Introduction

Aim

- Introduce techniques used with standard UNIX tools to perform basic Software Configuration Management functions in a multi-programmer, multi-product hardware and software development environment.

Background

- Transition from single product, single platform, single source tree environment to a mult-product, multi-platform, parallel development environment.

## Overview

State some problems

Offer some principles

Apply those principles

Mention some alternatives



## The Problems

Double Maintenance

- two (or more) "true" versions of the "same" source code

Shared Data

- attempting to share a single version causes instability during debugging

Simultaneous Update

- separate copies during debugging can lead to old bugs reappearing



## The Problems

Identification

- What version is this
  - — binary file?
  - — source file?
  - — document?

Reproducibility

- Can we reproduce a released binary from source after subsequent releases?

Variations

- Models of hardware
- Options





## The Problems

Environment

- What are `<sys/param.h>` and `-lc` today?

## The Principles

Common Baseline

- Prevent double maintenance
- Keep a history of the project

Private Playpens

- Avoid instability of shared data

Locking

- Prevent simultaneous update
- When to lock?
  - at checkout time?
  - just before checkin?
  - before integration testing!



## The Principles

Version Stamps

- Just a hint
- On all possible objects
  - executables
  - documents
  - arbitrary binaries can be difficult
- Not a substitute for a Configuration Definition



## The Principles

Configuration Definition

- For reproducibility
- A configuration is defined by
  - a list of every file used and its revision
  - the build procedure
  - the definition of the environment

Boxes

- Restrict build environment to that intended
- No unintended binding of
  `#include <sys/param.h>`
  `-lc`





## The Principles

Stability vs Currency

- Fundamental tradeoff
- Want stability while debugging
- Want currency while integration testing

Equity

- All objects are subject to the rules
- Wide interpretation of *source file*
  - documentation
  - memos
  - PLD equations

## The Techniques

SCCS supplies the primitives

- change control per file
- revision naming
- locking
- change history
- RCS would do just as well

Build on SCCS

- Wrappers
- Conventions



## The Techniques

Baseline is SCCS Tree

- 3D tree
  - depth
  - breadth
  - thickness?
- name translation required
- encourage revisions
- revisions are frozen once entered



## The Techniques

SCCS Wrapper

- `sccs` *command file ...*
- SCCS primitives, macros
  - `enter`
  - `get`
  - `lock`
  - `delta`
  - `diffs`
  - `sccsdiff`
- Name translation
  - `$SRCDIR`
  - `$SCCSDIR`





## The Techniques

Projects

- One per logical group of source files
  - of the same origin, eg. X11
  - comprising a common sub-system, eg. TCP/IP
- project is not the same as *product*
- exactly one baseline per project

## The Techniques

Playpens

- Private to user?
- Private to purpose
  - — bug fix, release build
- Directory tree paralleling baselines
- Built by selecting required revision of required files from one or more projects



## The Techniques

Parts Lists

- Lists every file, its revision and its project (baseline)
- Defines a configuration
- First step in build procedure
- *Configuration binding* is a manual process
  - — *getparts*
- Input to documentation
  - — *diffparts*
- cf. Bill of Materials



## The Techniques

Cross-environment

- originally for cross-compiling for other CPUs
  - — `xenv i960 make`
- isolates build process from vagaries of default environment
  - — `xenv svr4 make`
- may have multiple versions of "same" environment
  - — `xenv x11r3 make`
  - — `xenv x11r4 make`





## The Techniques

Bugfiler

- Central repository of reports
  - — bugs
  - — enhancement requests
  - — limitations, caveats, warnings
- Cross reference bug reports with SCCS revisions
- Advice to Support and Maintenance

## Alternatives

Cloned trees

- Yost, 1985
- Successive configurations are parallel trees
  - — linked to previous files where unchanged
  - — copied and edited where different



## Alternatives

Three Dimensional File System

- Korn and Krell, 1989
- Revisions are incorporated into file name space
- No need for special tools to access revisions
- file system has thickness as well as depth and breadth

Attributed File System

- *AtFS*, Mahler and Lampen, 1990
- File are objects with attributes



## Alternatives

Boxes

- Glew, 1989
- Use *chroot*(2) to enforce restricted environment

Ada Language System

- Babich, 1986
- Formalises inter-module dependencies (integrated *make*)
- An interesting example of the other approach





## Summary

In Favour

- Cheap
- Portable
- Acceptable
- Flexible

Against

- Lacks integration
- Lacks enforcement
- Flexible

## Bibliography

*The Mythical Man-Month*
F. P. Brooks, Jr, Addison-Wesley, 1975.

- Surprisingly relevant in 1991

*Software Configuration Management*
Wayne A. Babich, Addison-Wesley, 1986.

- Entertaining description of the problems and principles
- SCCS, RCS, make
- Ada Language System



## Bibliography

*Boxes, Links and Parallel Trees: Elements of a Configuration Management System*
Andy Glew, USENIX Software Management Workshop, April 3–4, 1989.

*CVS II: Parallelizing Software Development*
Brian Berliner, Winter 1990 USENIX Conference.



## Bibliography

*Integrating Configuration Management into a Generic Environment*
Axel Mahler and Andreas Lampen, SIGSOFT '90.

- Attributed File System, *AtFS*.

*The 3-D File System*
David G. Korn and Eduardo Krell, Summer 1989 USENIX Conference.

*The Cloned Tree Method of Revision Control*
David Yost, Summer 1985 USENIX Conference.

# Good Things Still Come In Small Packages

**Jack Dikian**
**Media-Lab Pacific**
***<jack@syd.sgi.oz.au>***

## ABSTRACT

It was ten years ago. It was yesterday. It was Version 7 from AT&T. It *is* Coherent from Mark Williams Co. Unix was small, simple and cheap. Unix is still small, simple and very, very cheap.

This paper takes a close look at the youngest and smallest kid on the block, and compares it with its older and bigger brothers, as well as its opponents. This is a review of Coherent Unix.

It was a long time ago, almost ten years ago in fact, when John Lions published what was a complete, annotated listing of AT&T's Version 7 kernel in the form of two booklets that together are no larger than this issue of AUUGN. Those documents, along with the University of New South Wales' "1980 Unix Companion" [UNSW], and "The C Programming Language" [R&K], more than anything else provided us with an extraordinary insight into Unix, the Unix philosophy, and its implementation language. For many, including myself, those documents were in a very real sense invaluable. Those documents, however, also represented something else; they were terse. The size of those notes made it possible to have them on hand virtually everywhere we went.

It is significant, therefore, that Coherent Unix is supplied with a single 1100 page manual that is on one hand very reminiscent of those early works, yet on the other borrows much from the more modern and accessible styles such as that found in the Kernighan-Pike "The Unix Programming Environment" [K&P]. The Coherent manual contains all the information that the user needs to install, use and, importantly, learn Unix. The manual covers the traditional Unix sections, namely the supported commands, system calls and subroutines, as well as excellent chapters providing tutorial like presentations. These include system administration, UUCP, awk, the C language, ed, lex, the m4 macro processor, make, MicroEMACS, text formatting, the shell and yacc.

I am placing special emphasis on the quality of the supplied manual for one very important reason; this variant of Unix potentially has a very ready market niche in the way of a low-end Unix training platform. Coherent also comes into its own through its use as a cost effective UUCP node or to provide the DOS user with an alternative vista.

"Coherent, A Multi-user, Multi-tasking Operating System For The IBM-PC/AT And Compatible 286 Or 386 Based Computers" [Coherent]. This brief product sketch printed on the cover jacket of the manual provides the PC enthusiast with enough flavour of Unix to encourage further curiosity. Coherent Unix comes from the 13-year-old compiler vendor Mark Williams Company, based at 60 Revere Drive, Northbrook, Illinois 60062 (uunet!mwc!sales). For US$99.95 you receive a 60-day money back guarantee Unix look-alike, and excellent user manual and free technical telephone support. This system is shipped on four 3.5 inch high density floppy disks and a single copy of the Coherent system manual. A registration card contains a nine digit serial number that the install program prompts for during the installation process.

The hardware requirements for this system are very modest when compared with even some DOS applications such as Microsoft Windows. The system requires an IBM AT or clone with 100% compatibility. It does not work on any of the MicroChannel platforms. One high density 3.5" or 5.25" floppy drive, a hard disk with at least 10MB free space, and a minimum of 640K RAM. The manual claims that the system will work with RLL, MFM and most ESDI disk controllers. It should also work with some SCSI host adapters. Coherent includes device drivers for line printers, HP laser printers, COM1 to COM4, RAM disks, tape drives, and the Adaptec SCSI disk controller. ESDI controllers include Ultrascope, Western Digital, and multiport from Arnet, Emulex and SEFCO. I suspect, however, that you need to take a close look at exactly what is and isn't supported. The release notes list more than 100 compatible systems, memory boards and disk controllers.

The preparation and installation took me approximately two hours to complete. In theory the actual Coherent install should only take about half an hour, depending on your CPU, but if you, like me, decide to partition the disk between DOS and Unix then you will need to backup your whole disk before you commence the installation. I carried out the installation on a very old 286 clone with 640K and a 40MB disk running DOS 4 with the Gemini EGA 2.4 BIOS. The provided install program drives the user through the installation process

from start to finish. It is no more difficult to install Coherent than it is to install any DOS application. Absolutely no prior knowledge of Unix is required. By far the trickiest section of the installation is when you are asked to re-partition the hard disk. Here you can nominate how much space you wish to allocate to Coherent and DOS, as well as defining the active partition. The operating system mounted on the active partition is booted automatically on start-up. The install program copes very well with the system it is being run on, and tries very hard to prompt you with specific and helpful messages as you go.

Once a partition has been allocated to Coherent, the install process bad blocks the nominated partition and makes the file system. You are now ready to reboot the system. The operating system on the active partition boots by default. If you load Coherent on the non-active partition, then you will need to press the number corresponding to the Coherent partition while the system is booting. If Coherent comes up OK, then the remaining three floppies are copied. This step takes a significant part of the overall installation process time. Uncompressing the man pages and the spell dictionary etc. is slow. Coherent with man pages and dictionary takes up 7MB. This leaves me 13MB of user file space on the Coherent partition, and a further 20MB DOS space. I should mention that the Norton Utilities [Norton] came in very handy at this point because the data remaining on the 20MB DOS partition was almost unusable. It took only minutes for Norton to make sense of the broken directories and help repair them.

Coherent Unix comes up multi-user after carrying out a rather slow (3 minutes for 7MB on 286) fsck and prompts for a login with "Coherent login:" At first you get the feeling that you are using a dumb terminal connected to a large AT&T SYS V rel 2 site. /bin looks quite comprehensive. But a closer inspection soon tells you why this is the small kid on the block. No POSIX compliancy, X-Windows or NFS. The C compiler is fast, but does not support medium and large models on the 286. Source code is not included, csh is not available, nor is off the shelf software.

Coherent does, however, fit into 640K of memory (it can address up to 16MB) with the kernel using up a whole 77K. It does give you text formatting facilities through nroff with ms. The manual also provides a 65 page chapter introducing nroff with very relevant examples. UUCP, as mentioned earlier, is supplied via uuinstall, uucp, uucico, uuxqt, uulog, uuname and uutouch. Once again, the large Remote Communications Utility chapter takes away a lot of the black magic from establishing uucp links. The public domain MicroEMACS is included, as is kermit. The stream editor sed, ed and elvis (vi) are well implemented. I especially found the yacc presentation and program examples quick to implement and easy to learn from. The C compiler, an assembler (for subroutines only), awk and the shell provide a well rounded development suite for training if not for developing real systems. No single platform supporting a dual operating system is complete without a data communication mechanism. Coherent provides a tar like utility called dos which allows the Coherent user to manipulate an MS-DOS file system. It can format or lable an MS-DOS file system, list the files in it, transfer files between it and Coherent or delete files from it. If you wish you can also buy a device driver toolkit for US$39.95.

Yes, there are other kids on the block. However Coherent is by far the best dressed for the price. I am going to take a quick look at three other products which Coherent contends with. The first is not really an operating system, but rather a suite of layered utilities called the MKS (Mortice Kern Systems) Toolkit. MKS sits on top of DOS and provides over 100 System V commands including the Korn shell and vi. However there are no development tools, and because of its dependence on DOS there is no multi-user/multi-tasking facilities. MKS costs $250.00. The second player is Minix (Mini Unix) from Prentice Hall, which is based on AT&T's Version 7 and is supplied, with source, on 12 3.5" floppy disks. Minix sits on the host hardware and requires at least a 10MB partition if source is to be included. Although there is no UUCP support, Minix does feature networking, rcp and Ethernet. The third, SCO (Santa Cruz Operation) XENIX [SCO] is really a heavy weight in features and price when compared Coherent. SCO has a 198K kernel and requires at least 1 to 2MB of memory and 30MB of disk. It costs $1495.00.

In conclusion, Coherent Unix from Mark Williams Co. is a truly high performance for value product. It combines the power and flexibility of Unix with the accessibility of PC based technologies. The manual is excellent and it alone is comparable to many speciality books costing many tens of dollars. The training sector is by far the most suitable environment for this product. Not only can this system be used to provide Unix concepts and training, but other areas such as C, shell, systems administration and text formatting can be mastered. The systems also lends itself as an ideal UUCP node.

The Mark williams Company claims it already has 10,000 satisfied users... make that 10,001.

**References**


[UNSW] *Unix Companion*, 1980, University of NSW.

[R&K] Kernighan, B. & Ritchie, D. *The C Programming Language*. Prentice Hall, 1978.

[K&P] Kernighan, B. & Pike, R. *The Unix Programming Environment*. Prentice Hall, 1984.

[Coherent] *Coherent Manual*, 1990, Mark Williams Co.

[SCO] Tech Specialist Journal, January 1991.

[Norton] *Norton Utilities, Advanced Edition 4.50*, 1987-1988, Peter Norton.

# USENIX News For AUUG Members

Donnalyn Frey is the USENIX Association Press Liaison. She provides members of the press, USENIX Association members, and AUUG members with information on the activities of the USENIX Association.

## 1. Winter 1991 Dallas Conference

The Winter 1991 Dallas USENIX Conference was held January 21 - 25, 1991 at the Grand Kempinski Hotel in Dallas, Texas. The theme of this conference was *What's next: by the year 2010, evolution or revolution? Unix derivative or Something Else?*

### 1.1 The Keynote Presentation

The keynote speaker at the conference was Flip Phillips, of the Pixar Animation Research and Development Group. Marc Donner wrote this review of the keynote presentation:

The keynote speaker at the Dallas conference was Flip Phillips, filling in for Eban Ostby, who was unable to attend due to illness. Phillips, an Animation Scientist (Technical Director and Animator) at Pixar, spoke from slides prepared by Ostby.

Phillips delivered a talk that was rich in technical detail. The overall message was that the techniques and skills required to produce and execute outstanding computer animations at Pixar are much the same as those required for good system programming. This was illustrated by a tour through the design of a collection of animation support tools developed at Pixar and used in their productions.

Phillips started off by briefly describing the original animation system developed at Pixar, a large and monolithic program that proved hard to modify and insufficiently general. Rather than continue to enhance this system, the people at Pixar decided to build a completely new system.

The new system was designed to be composed of multiple processes, taking advantage of the UNIX environment on which it was developed and run. In the process of describing the components, Phillips sought to demonstrate that the design and construction relied heavily on conventional system programming skills.

The animation support system designed at Pixar is built from seven subsystems. These subsystems include: file I/O, database, events, interactive I/O, graphics tools, animation tools, and modelling tools.

The 20 graphics tools are responsible for creating and maintaining an image on a display. The animation tools provide motion, and the modelling tools are used to orchestrate complete behaviors in the process of animation. The database system is the central repository of information concerning an animation, from which all of the tools receive parameters and into which they deposit information. The event handling system deals with the mouse, tablet, and other real-time activities.

Each of these components is required to run on all of the hardware platforms used at Pixar, including Sun Microsystems, CCI, and Silicon Graphics machines. This requirement presented significant challenges, since both BSD and System V interfaces were involved.

Some of the underlying system facilities that were required to implement this design include shared memory, interprocess communication, semaphores and other synchronization primitives, and a number of specialized languages with associated interpreters and compilers. In addition, a C language binding for graphics primitives, called RenderMan, was designed and implemented as part of the Pixar system.

The Pixar modelling system is based on a modelling language, designed by Ostby, that incorporates features of C, APL, and even *awk*. This language, in addition to vector operations based on APL primitives, includes concurrency primitives and articulated variables. An articulated variable is one that is varied externally to a routine during the routine's execution.

Shading and textures are produced with another specialized language. This language, also C-like, is used to generate surfaces procedurally. This technique has performance advantages over texture mapping, since no file operations are required when constructing a surface.

The quantity of data involved in animation and rendering is quite large and efficient algorithms for sorting are crucial to the task.

The talk ended with a showing of notable Pixar animations including short films and commercials: *Luxo, jr*, a Tropicana orange juice commercial, *Red's Dream*, an advertisement for Trident mint flavored chewing gum, *Tin Toy*, an advertisement for LifeSavers Holes candy, *Knick Knack*, and an advertisement for Listerine mouthwash. As Phillips noted, the films, such as *Tin Toy* win the Oscar awards and the commercials, such as LifeSavers Holes candy, pay the rent. All, however, are technically excellent.

### 1.2 The Program and Invited Talks

The program of the conference included:

- Kernels
- File system performance
- Threads and networks
- Interface tools
- Kernel panel
- File systems panel
- Programming tools File systems
- Objects in action
- Insecurity
- Distributed processing

and invited talks on:

- *Toolkit Graphics* by Doug Blewett of AT&T Bell Laboratories
- *Troff Macro Programming* by Sharon Murrel of AT&T Bell Laboratories and Jaap Akkerhuis of mt Xinu
- *UNIX Security Today and Tomorrow* panel, organized by Pat Bahn of GTE Government Systems and moderated by Bill Cheswick of AT&T Bell Laboratories
- *System Administration* by Rob Kolstad of Sun Microsystems
- *Using Distributed Objects* by Vinny Cahill of the University of Dublin
- *Debugging X and X Toolkit Applications* by Paul Kimball of Digital Equipment Corp.

### 1.3 The Terminal Room at the Conference

The USENIX Association hosted a Terminal Room with modems for a dialout connection and a T-1 connection to the Internet. Conference attendees could log onto their home or work systems to read their mail and contact other UNIX users directly from the conference. The USENIX Association thanks UUNET for the T-1 Internet connection, cisco, Digital Link, Graphon, NCD, Sun Microsystems, Talaris, Telebit, and Xylogics for the hardware that made the terminal room possible.

### 1.4 Best Student Paper Winner Garners Second Win

Margo Seltzer again won the Best Student Paper Award for her and Ozan Yigit's *A New Hash Package for UNIX*. Margo is a Ph.D. student at the University of California at Berkeley and Ozan is a software engineer at York University in Toronto.

## 2. Monograph Series on Advanced Computing Systems

The USENIX monograph series, to be published jointly with the MIT Press, is soliciting book length manuscripts for publication. The editor of the series, Marc Donner of IBM Research, and managing editor Alain Henon, are looking for monographs on languages, hardware, software, theory, and history in advanced computing. The editorial board is composed of Stuart Faulk of the Software Productivity Consortium, James Gosling of Sun Microsystems, Hank Levy of the University of Washington, Michael O'Dell of Bellcore, and Anne Rodgers of Princeton University.

To submit a manuscript or proposal for consideration for the Monograph Series, send a copy to:

Monograph Editor
USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710

or send electronic mail to monographs@usenix.org

## 3. Further Information on Conferences and Workshops

If you need further information regarding USENIX conferences or workshops, contact the USENIX Conference Office at

22672 Lambert Street
Suite 613
El Toro CA 92630 USA

Email to:

judy@usenix.org or
{uunet,ucbvax}!usenix!judy

Tel: +1 714 588 8649

FAX: +1 714 588 9706

## 4. Further Information about the USENIX Association

If you would like information on membership, or would like information on ordering USENIX publications (proceedings, manuals, *Computing Systems*, the Monograph Series, or the Association's newsletter, *;login:*, please contact the USENIX Association Executive Office at

2560 Ninth Street, Suite 215
Berkeley CA 94710 USA

Email to office@usenix.org

Tel: +1 415 528 8649

FAX: +1 415 548 5738

# Monograph Series on Advanced Computing Systems

The USENIX Association intends to publish books and monographs on the general topic of computing systems. The intended audience for these books is the community of system designers, builders, users, and scholars. Our intent is to publish material of lasting interest and importance, with an emphasis on actual systems. Subjects may include design, implementation, history, and analysis of real systems. While we are inspired by UNIX and UNIX-like systems, we do not expect to limit our attention to such systems in any way, as we see ourselves responsible to the entire systems community.

We see several specific needs that we would like to satisfy and for which we solicit manuscripts. The needs fall in two areas – books in traditional styles and formats about topics important to the systems community and things new or unusual.

Among things new or unusual, we are interested in exploring at least these ideas:

*Significant systems* – Many significant systems are documented, if at all, only in reference manuals or user guides. Journal publications often concentrate on narrow specific details, as is appropriate for focused technical audiences. What is lost is the broad description of the design and its evolution, with consideration of the success and failure of specific features and lessons learned.

*Code* – We are interested in exploring the possibilities of publishing code to read. A truism among the programming community is that one learns to write good programs by reading good and bad programs. Sadly, there is little code available to read. The recent interest in public-domain code and open systems has increased the quantity of high-quality source code available. Many open questions in the publication of code remain to be explored. The conventional codex form, long accepted as appropriate for literary works and texts, may not be the right one for programs. Very few experiments have been made with this form, something that we hope to encourage. The audience for published code includes serious students of systems, including both the undergraduate and advanced levels, and practitioners involved with development, modification, and analysis of actual systems.

*Important technical reports* – Many important technical reports, issued in small numbers by industrial organizations, research labs, or university departments, are not disseminated as widely as they merit. This is often because the originating organization doesn't have the resources or the will to publish them more widely and because the material is deemed inappropriate by commercial publishers because of its narrow scope or limited size. Many technical reports are too large for journal publication and too small for conventional book publication. We hope to provide a means of publication and distribution of the best of these.

Authors will enter into a contractual arrangement with USENIX. The Association is in the process of selecting a publisher to handle marketing and distribution. We hope that you will consider this arrangement a viable option for your next manuscript.

To submit a manuscript or proposal for consideration for the Monograph series, send a copy to:

Monograph Editor
USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710

or send electronic mail to

monographs@usenix.org.

# Book Review

## UNIX System Administration Handbook

**Evi Nemeth, Garth Snyder, and Scott Seebass**

(Prentice-Hall, 1989, ISBN 0-13-933441-6)

*Reviewed by Nichlos H. Cuccia*

cuccia@Sybase.com

UNIX system administration is a topic about which relatively little has been written, especially when compared to topics such as shell programming and document formatting. Much of what has been written until recently has been obsolete for many years, written with a decided slant towards AT&T's System V, or focused on specific topics such as writing *termcap* or *terminfo* entries, or managing UUCP or netnews systems. While this may be adequate for a system administrator responsible for a handful of machines running XENIX or System V, it ignores the needs of system administrators who have to manage networks of machines running BSD or BSD-based UNIX. It is into this gap that Evi Nemeth, Garth Snyder, and Scott Seebass leap with the *UNIX System Administration Handbook.*

The *Handbook* is written at a level suitable for beginning systems administrators. It covers a wide range of system and network administration issues, including: booting and shutting down systems; the use of superuser privileges; filesystem organization and file permissions; monitoring and controlling processes; adding and deleting new users; adding terminals and devices, installing device drivers, and configuring system kernels; managing AT&T- and BSD-based printing systems; network hardware, software, and management issues; electronic mail, UUCP, netnews, and sendmail; backups and restorations; system daemons and processes spawned by *cron*; quotas and per-process limits; system security concerns; and issues such as file revision control, local documentation, reporting bugs, and disk cleanup. Its seventeen appendices include C and *csh* source code for several useful utilities, in addition to a sample *sendmail.cf* file, a makefile used for saving and restoring system configuration files to or from tape, and forms for domain, IP address, and UUCP site registration.

The *Handbook* is strongest when it discusses, in detail, topics that may mystify beginning systems administrators. Sections of the book worthy of notice are: chapter two, which takes the reader through a sample */etc/rc* file, as well as delineating strategies for dealing with systems that won't boot, bad boot media, or corrupt filesystems; chapter fourteen, which discusses network issues ranging from the hardware required to set up a network and the configuration of system software to administrative issues including network security, licensing, and tools for monitoring and debugging network software and hardware; and chapter fifteen, which discusses *sendmail* in great detail.

The *Handbook* does have its flaws, however. There is a tendency to list online sources of information or source code as anonymous *ftp* repositories. This is reasonable for sites on the Internet, but the omission of anonymous *uucp* sites limits the ability of UUCP-only sites to acquire such software as the latest Berkeley *sendmail* or BIND. The omission of the semicolon after the closing bracket of the tests in the sample */etc/rc* file – an omission that does not occur in the snippets from an */etc/rc.local* file in chapter fourteen – could have dire consequences for any user who tries to use it on a production system. These flaws, however, are not serious enough to diminish the usefulness of the *Handbook.*

In short, the *UNIX System Administration Handbook* is a must-have item for any user who wants to learn how to administer UNIX systems. Its occasional humorous touches, cute but informative drawings (especially the one that shows the difference between a bug and a feature), and its historical anecdotes (such as the origins of the name of the *biff* command) make it an interesting read.

# An Update on UNIX and C Standards Activity

## May and June, 1990

*Jeffrey S. Haemer*
Report Editor, USENIX Standards Watchdog Committee

### Report on USENIX Standards Watchdog Committee

Jeffrey S. Haemer <jsh@ico.isc.com> reports on spring quarter standards activities:

#### What these reports are about

Reports are done quarterly, for the USENIX Association, by volunteers from the individual standards committees. The volunteers are familiarly known as *snitches* and the reports as *snitch reports.* The band of snitches and I make up the working committee of the USENIX Standards Watchdog Committee. Our job is to let you know about things going on in the standards arena that might affect your professional life – either now or down the road a ways.

We don't yet have active snitches for all the committees and sometimes have to beat the bushes for new snitches when old ones retire or can't make a meeting, but the number of groups with active snitches continues to grow (as, unfortunately, does the number of groups).

We know we currently need snitches in 1003.6 (Security), 1003.11 (Transaction Processing), 1003.13 (Real-time Profile), and nearly all of the 1200-series POSIX groups, There are probably X3 groups the USENIX members would like to know about that we don't even know to look for watchdogs in. If you're active in any other standards-related activity that you think you'd like to report on, please drop me a line. Andrew Hume's fine report on X3B11.1 is an example of the kind of submission I'd love to see.

If you have comments or suggestions, or are interested in snitching for any group, please contact me (jsh@usenix.org) or John (jsq@usenix.org). If some of the reports make you interested enough or indignant enough to want to go to a POSIX meeting, or you just want to talk to me in person, join me at the next set, July 16-20, at the Sheraton Tara, in Danvers, Massachusetts, just outside of Boston.

The USENIX Standards Watchdog Committee also has both a financial committee – Ellie Young, Alan G. Nemeth, and Kirk McKusick (chair); and a policy committee – the financial committee plus John S. Quarterman (chair).

An official statement from John S. Quarterman, USENIX Standards Liaison:

The basic USENIX policy regarding standards is:

> to attempt to prevent standards
> from prohibiting innovation.

To do that, we

- Collect and publish contextual and technical information such as the snitch reports that otherwise would be lost in committee minutes or rationale appendices or would not be written down at all.
- Encourage appropriate people to get involved in the standards process.
- Hold forums such as Birds of a Feather (BOF) meetings at conferences. We sponsored one workshop on standards, and are cosponsoring another in conjunction with IEEE, UniForum, and EUUG. (Co-chairs are Shane P. McCarron, ahby@uiunix.org, and Fritz Schulz, fritz@osf.osf.org. Contact them for details.)
- Write and present proposals to standards bodies in specific areas.
- Occasionally sponsor White Papers in particularly problematical areas, such as IEEE 1003.7 (in 1989).
- Very occasionally lobby organizations that oversee standards bodies regarding new committees, documents, or balloting procedures.
- Starting in mid-1989, USENIX and EUUG (the European UNIX systems Users Group) began

sponsoring a joint representative to the ISO/IEC JTC1 SC22 WG15 (ISO POSIX) standards committee.

There are some things we do *not* do:

- Form standards committees. It's the USENIX Standards Watchdog Committee, not the POSIX Watchdog Committee, not part of POSIX, and not limited to POSIX.
- Promote standards.
- Endorse standards.

Occasionally we may ask snitches to present proposals or argue positions on behalf of USENIX. They are not required to do so and cannot do so unless asked by the USENIX Standards Watchdog Policy Committee.

Snitches mostly report. We also encourage them to recommend actions for USENIX to take.

---

## Report on IEEE 1003.0: POSIX Guide

Kevin Lewis <klewis@gucci.dco.dec.com> reports on the April 23-27 meeting in Salt Lake City, UT:

### Where we are

The Utah meeting of the IEEE 1003.0 working group marks the beginning of its third year. Let's step back for a moment to review the past two. We have gone from scratch to a 180-page document, the content of which represents about 70% of the content goal that we set for our work two years ago. (More on this in a moment.)

This effort represents the contributions of a core group of 15 to 18 people. In 1988, 14 vendor organizations and 16 user organizations were represented within the group. Today, we have nine vendor organizations and 16 user organizations represented. Of course, the only official formal organizational representatives allowed within IEEE working groups are accredited institutional representatives (currently Usenix, UniForum, X/Open, Unix International, and the Open Software Foundation each supply one to the POSIX effort), but that does not stop me from checking the sign-up sheet whenever a new face shows up, to see where he or she works. For example, I think someone from the Univ. of Berkeley involved in BSD UNIX development has a vendor's perspective, while I place attendees from NIST and the Air Force in the user category because I believe they focus on the interests of their own end users. Our stable steady user representation is essential: our ultimate targets are users trying to walk through the POSIX maze.

The 70% completion of our initial content goal includes the introduction of the "profile" concept, which has led to increased activity within the IEEE TCOS Standards Subcommittee to create groups to define profiles (which may be good or bad depending on your own prism). The concept of profiles is also part of the US's contribution to the ISO community, made through its participation in the JTC1 Technical Study Group on Application Portability (JTAP), within which the "profiles" concept has now garnered wide acceptance.

"What is a profile?" you ask. Users seeking open system solutions need to know what parts of the open system environment (OSE) address their requirements. If a user could reach into the full basket of OSE parts and pull out only those he or she specifically needs, those selected parts would be his or her application environment profile. What should the user do if he or she needs something not in the basket? Come to our next meeting with a recommendation. [Editor: Or drop Kevin a line, or post something to comp.std.unix!]

### Where we're going

Dot Zero still faces hard decisions in two areas:

1. the necessity or desirability of parts of our guide. (Two parts that I very much think are candidates for this discussion are User Interface and Security.)

2. The final bounds of the profile concept/definition.

The group's arguments in these areas are not frivolous, but if they continue much longer, the resulting lack of movement will hurt our overall effort.

I came out of this meeting feeling that everyone is committed to getting over these hurdles soon (i.e., by the July meeting). Our

chair, Al Hankinson, has also stated that we should target December 1990 for a mock ballot. I wholeheartedly agree. This will add the impetus that we need. Let's see if we have the self-discipline to get there.

## Report on IEEE 1003.1: System services interface

Paul Rabin <rabin@osf.org> reports on the April 23-27 meeting in Salt Lake City, UT:

#### Introduction

The POSIX 1003.1 working group is the oldest POSIX group, responsible for specifying general-purpose operating system interfaces for portable applications. This group developed the original 1988 POSIX standard, and is now responsible for writing supplements and revisions to that standard. This work includes

- corrections and clarifications to the 1988 POSIX standard
- material that was too controversial to handle before
- new interfaces requested by other POSIX working groups

Like other working groups developing "base standards," the 1003.1 working group is responsible for writing both C language and language-independent versions of the specifications that it develops. So far, the group has concentrated on the C language versions, but there is increasing pressure to make progress on the language-independent specifications.

The working group recently completed a revision of the 1988 POSIX standard, and is currently working on a supplement to that revision.

There has been a lot of turnover in the group since the 1988 POSIX standard was completed, but there are still a few old-timers to provide continuity. About 15 people attended the last two meetings. This seems to be a good size for getting work done. This is definitely a technical crowd; check your politics at the door.

For more information about the group and how to participate, contact the chair, Donn Terry, at donn@hpfcla.fc.hp.com or hplabs!hpfcla!donn. Send comments and proposals to the secretary, Keith Stuck, at keith@sp7040.uucp. I've made this report a bit more detailed than usual in order to give the technical details wider exposure. New proposals and comments on any of the current active proposals or issues are welcome.

#### 1003.1a Status

1003.1a is the recently completed revision to the 1988 POSIX standard. No new interfaces or features were introduced, but the text was revised in several ways. The main reason for the revision was to prepare the text for balloting as an ISO standard, so the document had to be made to look like an ISO standard. This meant adding ISO boiler-plate, changing external document references to pretend that only standards exist, changing internal cross-references so that chapters are renamed sections, and sections are renamed clauses and subclauses, changing "will" to "shall," etc., ad nauseam. While the working group was having fun with all that, they took the opportunity to do some cleaning up. They corrected errors, clarified unclear language, and changed the function synopses to use ANSI C prototypes. The group did make one normative change, which was to specify reserved namespaces. This will allow implementations and revisions to the standard to define extensions without breaking existing conforming applications. It's messier than you might think.

After four recirculation ballots, IEEE balloting was completed in April. Now it has to get through the ISO balloting process. See the recent snitch report on 1003.5 for a description of how IEEE and ISO balloting is synchronized, and what all of the acronyms mean.

ISO has been balloting 1003.1a as ISO/IEC DIS 9945-1. After the first ISO ballot, JTC1 approved 9945-1 for promotion to full IS status. This approval was overruled by the ISO Central Secretariat on the grounds that the document format was still not satisfactory (still haven't caught all of those "will"s). Rather than publish the current document and then immediately revise, ballot, and publish it

again, it was decided to create a new DIS and to start a second round of ISO balloting. This will cause a delay in the publication of the international POSIX standard (and hence also the IEEE POSIX.1 standard). The U.S. Technical Advisory Group (TAG) is responsible for generating the U.S. ballot. Assuming that no normative changes are introduced by the ISO balloting process, the resulting document will be published by IEEE as IEEE Std 1003.1-1990.

**1003.1b Status**

Since 1003.1a is now out of IEEE's hands, the working group spent the Utah meeting working on 1003.1b, the first supplement to 1003.1a. This will include some corrections and clarifications that didn't make it into 1003.1a, but will mainly consist of new interfaces and features.

1003.1b has been in the works for several meetings, so the draft already contains a lot of material. The first day was devoted to revision of the draft, the rest of the week to considering new proposals. The previously announced schedule for 1003.1b specified the Utah meeting as the cutoff date for new proposals. Unfortunately, some expected proposals were not received, and some that were received were not ready for incorporation, so the cutoff was deferred until the next meeting, in Danvers, Massachusetts.

Draft 2 of 1003.1b was distributed just before the meeting in Utah. Draft 3 should be available before the Danvers meeting. 1003.1b is expected to be approved sometime in early 1991, and to be published by IEEE as a separate supplement to the IEEE Std 1003.1-1990.

**New Features in the Current Draft of 1003.1b**: Draft 2 of P1003.1b includes a new data interchange format, and new interface specifications for symbolic links, environment list access, and file-tree walking. These had been proposed and generally accepted at previous meetings. Many new issues were raised and discussed.

*Symbolic Links*: P1003.1b adds BSD symbolic links to the 1988 POSIX standard as a new required feature. New interfaces for *symlink()*, *readlink()*, and *lstat()* are specified, and the definition of pathname resolution is amended to include the handling of symbolic links. Implementations may optionally enforce a limit on the number of symbolic links that can be tolerated during the resolution of a single pathname, for instance to detect loops. The new symbol {*_POSIX_SYMLOOP*} is defined to be the minimum value of such a limit. A new error, *[ELOOP]*, is returned if such a limit is exceeded. Symbolic links that are encountered in pathname prefixes are always resolved. Symbolic links named by the final component of a pathname will be resolved or not, depending on the particular interface. By default, such symbolic links will be resolved, unless specified otherwise. The interfaces that will not resolve symbolic links named by pathname arguments are:

*readlink()* If the pathname argument names a symbolic link, the contents of the link will be returned.

*lstat()* If the pathname argument names a symbolic link, a stat structure will be returned for the link itself.

*unlink()* If the pathname argument names a symbolic link, the link itself will be removed.

*rmdir()* If the pathname argument names a symbolic link, the link will not be followed and the call will fail.

*open()* Symbolic links are followed, unless both *O_CREAT* and *O_EXCL* are set. If both *O_CREAT* and *O_EXCL* are set, and the pathname argument names an existing symbolic link, the link will not be followed and the call will fail.

*link()* If the *new* pathname names a symbolic link, the link will not be followed and the call will fail. If the *old* pathname names a symbolic link, the link will be followed. This is the BSD behavior. SVR4.0 does not follow the link in this case, thus supporting hard links to symbolic links. The working group felt that the SVR4 behavior unnecessarily restricts implementations (for instance, those that do not implement symbolic links with inodes), and has much more complex semantics.

*rename()* If the *old* pathname names a symbolic link, the link will not be followed. Instead, the symbolic link itself will be renamed.

If the *new* pathname names a symbolic link, it will not be followed. Instead, the symbolic link will be removed, and a new hard link will be created naming the file that was previously named by the *old* pathname.

The 1988 POSIX standard specifies that if the *new* pathname names an existing file, *rename()* will fail if the *new* and *old* pathnames do not either both name directories or both name non-directory files. This rule needs to be expanded to include the case of the *new* pathname naming a symbolic link. Should the *rename()* call fail depending on whether or not the symbolic link named by the *new* pathname itself names a directory or a non-directory file? This will be resolved at the next meeting.

Symbolic links are not required to have any attributes other than their file type and their contents. This is intended to provide the simplest semantics and to allow the greatest latitude for implementations.

*Other BSD Interfaces*: P1003.1b also includes specifications for the following interfaces:

*fchmod()*
*fchown()*
*fsync()*
*ftruncate()*

*Environment List*: The ANSI-C standard defines the *getenv()* function to retrieve the value corresponding to a given name in a program's environment list, but does not specify the implementation or initialization of that list. The 1988 POSIX standard specified the traditional list implementation using the external variable *environ*, and specified the initialization of the list by the *exec* functions. In an attempt to extend the set of high-level interfaces to the environment list, and to pave the way for the possible eventual removal of *environ*, the working group has included *putenv()* and *clearenv()* interfaces in 1003.1b. Three problems have been identified with these high-level interfaces:

1. They cause static data to be shared between the application and the implementation. Neither the application nor the implementation can easily manage the storage for environment "*name=value*" strings.

2. They are not robust. The interactions between the high-level interfaces and access via *environ* are not specified.

3. They can't be easily extended to handle multiple lists. There is no way to copy a list, or to build a new list for *execle()* or *execve()*.

The *putenv()* and *clearenv()* interfaces may be removed from 1003.1b at the next meeting if a revised proposal does not appear.

*File Tree Walk*: The 1003.1 working group promised the 1003.2 group (Shell and Utilities) that a mechanism would be provided for walking a directory tree of unbounded depth using any given (non-zero) number of free file descriptors. The Berkeley folks have implemented a set of high-level interfaces defined by David Korn of Bell Labs, and proposed them for inclusion in 1003.1b. These interfaces support every option and search order required by the 1003.2 commands. The 1003.1 group wants a simpler interface suitable for typical application programs, so Keith Bostic will put the proposal on a "weight-reducing diet" and resubmit it for the next draft.

The high-level file-tree walk interfaces can be implemented using only the existing 1003.1 interfaces. Since 1003.1 does not define a portable way to save and restore file position for a directory and cannot hold a file descriptor open for each directory level, the implementation must read and save all directory entries each time a new directory is visited. This requires only a single file descriptor (or whatever resource is used by by *opendir()*). If the underlying system does provide a way to save and restore file position for directories, the file-tree walk implementation can use it to reduce memory consumption.

There was a discussion about whether it is possible (and preferable) to improve the low-level directory interfaces instead of adding new high-level interfaces. Do the high-level interfaces really add new functionality for portable applications? Do they belong with the low-level operating system interfaces specified in 1003.1?

Walking an unbounded file tree requires an unbounded number of directory file positions to be supported using a bounded number

of file descriptors. Either *seekdir()* and *telldir()* are needed, or an unbounded number of *opendir()*s must use a bounded number of file descriptors. The working group has already rejected *seekdir()* and *telldir()* because they cannot easily be supported on implementations that use a non-linear directory format. A prohibition of simple implementations of *opendir()* using file descriptors is also likely to be rejected.

The underlying problem is that the orderedness of directory entries was implicit in the traditional implementations, but was not made fully explicit in 1003.1, partly out of a desire to permit alternate implementations (for instance, b-trees). As a result, orderedness must now be imposed by the application. On a non-linear directory implementation, if positioning is not supported, even *opendir()* must read in the whole directory.

*Data-Interchange Format*: The 1988 POSIX standard specified two data-interchange formats based on existing utilities. These define the data-stream encoding of a sequence of files, together with their pathnames and other attributes. The first format is based on *tar* and encodes files as a stream of 512-byte blocks. The second format is based on *cpio* and encodes files as an unblocked byte stream.

The ISO POSIX group (JTC1/SC22/WG15) pointed out that both of these formats are incompatible with accepted international and U.S. standards. After some arm twisting, the 1003.1 working group agreed to devise a new data interchange format based on IS 1001: 1986, which is more or less equivalent to ANS X3.27-1987, the familiar ANSI labeled tape format.

The current draft of 1003.1b includes the framework for the new specification, but a lot more work is needed. Previous meetings discussed alternate proposals. The topic has been strangely quiet lately, considering the confusion that may be expected when it goes to ballot. It wasn't discussed at the Utah meeting at all.

{_POSIX_PATH_MAX}: **A Clarification**: The 1988 POSIX standard included two conflicting statements regarding {*_POSIX_PATH_MAX*} and {*PATH_MAX*}: one said that the null was included in the count; the other said that the null was excluded. Traditional implementations have included the trailing null; some new implementations have excluded the null.

One alternative or the other had to be endorsed. The working group decided that {*_POSIX_PATH_MAX*} should not include the trailing null, since specifying this will not break currently conforming applications.

**Headers and Name-Space Control**: Since 1003.1b is adding many new identifiers to the standard, there was discussion about whether new identifiers should be declared in new headers, or whether existing headers could be used, with new feature-test-macros to control visibility of the additional identifiers. It was agreed that although both headers and feature-test macros control identifier visibility, their functions are complementary. Headers are appropriately used to divide name-spaces horizontally, by functionality. Feature-test macros are appropriately used to divide name-spaces vertically, by specification level.

With this understanding, the group decided that new identifiers will be declared in the "right place." A new header will be created only if no existing header is functionally appropriate.

A new feature-test macro will be specified by 1003.1b and subsequent revisions: *_POSIX_1_SOURCE*. This macro takes ordinal values, starting with 2 for 1003.1b, and will be incremented by 1 for every subsequent revision. If the value is 1, the effect will be the same as if *_POSIX_SOURCE* were defined.

There are two changes here. The new name was used to indicate that the macro only controls the visibility of identifiers defined in POSIX.1. The usage was changed to allow the value to indicate the particular revision or supplement to the standard, rather than having to create a new macro each time. This should simplify the construction and maintenance of header files.

**Requests**: Two requests were made by vendors trying to support POSIX behavior on non-*UNIX* file systems:

1. that {*_POSIX_LINK_MAX*} be reduced from 6 to 2

2. that {*_POSIX_PATH_MAX*} be reduced from 255 to 252

Both requests were rejected. Either of these changes could have made existing conforming applications non-conforming. Even where the risk of breaking applications seemed small, the working group was reluctant to set a precedent without a pretty good rationale to protect them against similar requests in the future.

**New Proposals**: Five proposals for new interfaces were submitted for inclusion in 1003.1b, many of which provoked lively discussion. Some were accepted, some were rejected, and others were deferred to allow a revised proposal to be submitted or to allow more time for consideration.

*seteuid(), setegid()*: Bob Lenk and Mike Karels proposed a set of changes to the way the effective user and group id's are handled, in order to provide better support for *setuid/setgid* programs.

1. Require that all implementations support the functionality of the saved user ID and saved group ID. These process attributes are set by the *exec* functions and by privileged calls to *setuid()* and *setgid()*.

2. Add *seteuid()* and *setegid()* as new functions that change only the effective user ID and effective group ID, respectively. A change is allowed if the proposed new user/group ID is the same as either the real user/group ID or the saved user/group ID.

3. Redefine the {*_POSIX_SAVED_IDS*} option to apply only to non-privileged calls to *setuid()* and *setgid()*.

This proposal has general support in the working group, and will be included in the next draft of 1003.1b.

The discussion of this proposal led to a general lament about how unclear the group model is in the 1988 POSIX standard, perhaps the result of a hasty marriage between the System V and BSD models. At the next meeting, the working group intends to add new text to P1003.1b to clarify the relation between the effective group ID and the supplementary group list.

*Magnetic Tape Support*: The 1003.10 working group (Supercomputing Profiles) proposed new interfaces to support basic controller functions for magnetic tape drives, based on the *ioctl()* commands supported in 4.3BSD. Although support for these interfaces would be optional in 1003.1b, the working group decided that the functions should be further specified according to whether they are:

1. required for all types of tape drives;
2. required only for 9-track tape drives;
3. required only for cartridge tape drives; or
4. optional on all types of tape drives.

The proposal needed further revision, but was generally supported by the working group.

The submitted proposal also included interfaces for mounting labeled tape volumes. These were considered to be inappropriate for inclusion at this time and will be deferred until a later revision of the standard.

*Checkpoint/Restart*: The 1003.10 working group also proposed new (optional) interfaces for checkpointing and restarting processes. This proposal is based on two existing implementations. The interfaces are intended to protect very-long-running applications from both scheduled shutdowns and unexpected failures of the system.

The 1003.1 working group was not happy to have to deal with this and had lots of questions. Were programming interfaces for portable applications *really* needed, or was a command interface sufficient? How much state needed to be saved in the checkpoint? What if the processes depended on additional state information that was not in the checkpoint, such as file data or the states of other communicating processes or devices? In this case, the restart would only be successful if this additional state had not changed since the checkpoint. How could such changes be detected or prevented? What is the set of interfaces that an application can use and be sure that it can be checkpointed and restarted successfully, without dependencies on additional state? Should applications have a mechanism for checkpointing themselves, or for blocking an external request that they be checkpointed?

Because a checkpoint/restart mechanism will have a major impact on implementations, and the requirements are not yet clear, the working group was unwilling to endorse the current proposal. A task force made up of representatives of the 1003.1 and 1003.10 working groups will be created to clarify the requirements and revise the proposal.

This proposal is going to need a lot more discussion, so checkpoint restart interfaces will almost certainly not be included in P1003.1b, but they may be adopted in a subsequent revision.

*Messaging*: The UniForum proposal for new messaging interfaces has been before the 1003.1 working group for a couple of meetings now. The proposed interfaces are intended as replacements for the message catalog interfaces specified in XPG3, and differ from those in several ways (since the discussion was fairly contentious, I'll try to be objective):

1. The XPG3 interfaces identify a message by the triple: <catalog name, set ID, msg ID>, where catalog name is a file name, and set ID and msg ID are integers. The UniForum interfaces identify a message by the triple: <locale name, domain name, message name>, where locale name, domain name, and message name are all strings. The locale for messages is specified by the new *LC_MESSAGES* category of the locale. Advocates of the UniForum proposal claim that string identifiers are easier to use and are more robust against errors during application development and maintenance.

2. In the XPG3 scheme, each message catalog is an ordinary file. Message catalogs must be specified by filename and explicitly opened before messages can be retrieved. The *NLSPATH* environment variable provides a search path for message catalogs that can be parameterized by (among other things) the language, territory, and codeset fields of the *LANG* environment variable. In the UniForum scheme, groups of messages are specified by an abstract "domain." A default domain can be set to control message accesses, or the domain can be explicitly specified for an individual message access. Advocates of the UniForum proposal claim that the binding of message catalogs to files unnecessarily restricts implementations and imposes a more complex interface on application developers.

3. The XPG3 interface includes an additional string argument that is returned in case no message specified by <set ID, msg ID> can be retrieved from the message catalog. In the UniForum proposal, the message name itself is returned if the message cannot be found. Advocates of the UniForum proposal point out that the message name string makes a separate, "default" message string unnecessary.

In addition, the UniForum proposal includes a specification of the message source file format that differs from the format specified in XPG3.

1. In the XPG3 format, message strings are implicitly delimited: at the beginning by the preceding message ID followed by a single space or tab character, and at the end by an unescaped newline. In the UniForum format, all strings, including domain names, message ID's, and message strings, are explicitly delimited by double quotes ("). Adjacent strings separated by white-space characters are concatenated. Advocates of the UniForum proposal claim that the new format provides better support for multi-line strings and for leading and trailing white-space characters in strings.

2. In the XPG3 format, the message ID and its corresponding message string are implicitly determined by their position on a source line. In the UniForum format, explicit directives are provided for message ID's and message strings. Advocates of the UniForum proposal claim that the new format is extensible. New attributes may be added to message entries, such as screen coordinates or font size.

3. The XPG3 format includes directives for deleting individual messages and sets of messages, and the associated *gencat* utility takes no switches. In the UniForum proposal, all deletion semantics are provided by switches on the associated *gentext* utility.

There was much discussion of the interfaces; less about the source file format. The most divisive issue was whether string message IDs are preferable to numeric message IDs.

Among those who felt that the new interfaces are better, there was disagreement about whether the advantages outweighed the cost of conversion for applications and implementations based on the XPG3 interfaces. The rationale accompanying the UniForum proposal described several ways to convert applications from the XPG3 interfaces to the proposed new interfaces.

The working group asked X/Open to submit the XPG3 messaging interfaces as an alternate proposal, since they represent existing practice, and X/Open has agreed to do so. X/Open has said that they will follow POSIX if POSIX endorses a different interface. The decision regarding which, if any, messaging proposal to include in 1003.1b will be made at the POSIX meeting in Danvers.

It's hard to predict the fate of this proposal. The UniForum proposal represents the consensus of one of the leading internationalization working groups and is reported to have some support within X/Open. On the other hand, the POSIX working groups are obliged to respect existing practice. Watch this space.

*/dev/stdin, /dev/fd/***n**, *etc.*: There was an unofficial proposal from members of the 1003.2 working group that *open()* be extended to recognize the special strings `/dev/stdin`, `/dev/stdout`, `/dev/stderr`, and `/dev/fd/N`, and return a new file descriptor *dup()*ed from *STDIN_FILENO*, *STDOUT_FILENO*, *STDERR_FILENO*, or file descriptor *N*, respectively. This proposal was intended to allow simplified command line parsing, by eliminating special casing for "-" and "--" arguments. The proposal was rejected after a short exploration of the possible semantics of these pathnames when used with *link()*, *rename()*, etc.

#### Conclusion

As you can see, there's a lot going on. Even though most of the attention has shifted to other working groups, the 1003.1 group is busy revising and extending the 1988 standard. The group is small now, by POSIX standards, but their work is as important as ever.

## Report on IEEE 1003.2: Shell and tools

Randall Howard <rand@mks.com> reports on the April 23-27 meeting in Salt Lake City, UT:

#### Background on POSIX.2

The POSIX.2 standard deals with the shell programming language and utilities. Currently, it is divided into two pieces:

- POSIX.2, the base standard, deals with the basic shell programming language and a set of utilities required for *application portability*. Application portability essentially means portability of shell scripts and thus excludes most features that might be considered interactive. In an analogy to the ANSI C standard, the POSIX.2 shell command language is the counterpart to the C programming language, while the utilities play, roughly, the role of the C library. POSIX.2 also standardizes command-line and function interfaces related to certain POSIX.2 utilities (e.g., popen, regular expressions, etc.). This document is also known as "Dot 2 Classic."

- POSIX.2a, the User Portability Extension or UPE, is a supplement to the base POSIX.2 standard; it will eventually be an optional chapter of a future draft of the base document. The UPE standardizes commands, such as screen editors, that might not appear in shell scripts but are important enough that users must learn them on any real system. It is essentially an interactive standard that attempts to reduce retraining costs incurred by system-to-system variation.

Some utilities have interactive as well as non-interactive features. In such cases, the UPE defines extensions from the base POSIX.2 utility. An example is the shell, for which the UPE defines job control, history, and aliases. Features used both interactively and in scripts tend to be defined in the base standard.

Together Dot 2 Classic and the UPE will make up the International Standards Organization's IS 9945/2 — the second volume of the proposed ISO four-volume standard related to POSIX.

In addition to providing current information about the activities of the Working and Balloting Groups for POSIX.2, a special topic of focus will be chosen for each report. Therefore, the reader is referred to earlier reports for information on such topics as the history of the *Shell Wars* and the controversial scope of the UPE. The next section talks about the functions, rather than utilities that are found with POSIX.2.

**The POSIX.2 API Functions**

Perhaps it will come as a surprise to many readers that the POSIX *Shell and Utilities* standard also contains specifications for about fourteen new or extended C function bindings – in effect, its own API extending the POSIX.1 bindings – as follows:

*confstr(), sysconf()* The first function was created to provide string-valued configuration-specific values such as the default setting of the PATH environment variable. The second extends the POSIX.1 function of the same name with numeric-valued configuration information such as the largest scale value in the *bc* utility and the implementation's line length restriction.

*fnmatch()* This functional interface implements the form of pattern matching used by file-name generation (*glob*) in the shell, *case* statements in the shell, and the `-name` option of the *find* utility.

*getopt()* This functional interface provides a standard utility argument parser that enforces the "standard utility syntax" guidelines and might be used to implement the *getopts* utility from POSIX.2.

*glob(), globfree()* This set of functions does shell-style file-name generation and presumably calls the *fnmatch()* function.

*popen(), pclose()* This pair of functions, which are a part of the standard I/O package on conventional UNIX systems, provides the ability to communicate through pipes to another process by executing a string in the POSIX.2 shell language.

*regexec(), regcomp()* This set of routines provides support for both the *Basic* and *Extended* Regular Expressions defined in POSIX.2, including full internationalization support.

*wordexp(), wordfree()* This set of routines provides a mechanism for an application to use word expansion (parameter expansion) that is compatible with the POSIX.2 shell language. Although most implementations of this routine will normally call the shell, it is (at least conceptually) possible that the shell be implemented to call these routines for word expansion.

*system()* This "classical" function executes a command written in *shell language*.

All of these functions form part of an optional C binding to POSIX.2 and it is expected that the soon-to-be-released, draft version of the NIST FIPS will make this "optional" functional interface mandatory for US government procurements. Other language-binding working groups, such as those exploring Ada and FORTRAN, are presumably encouraged to add their own optional bindings if they so wish.

Although the inclusion of these functions seems to be a little out of place in a shell-and-tools standard, there is some rationale for this. In fact, when POSIX consisted only of POSIX.1, the early attempts to define *system()* and *popen()* made apparent the need to completely specify the *shell language* in which the argument string to these functions was written. That, in turn, along with the desire to standardize the classical UNIX utility set, led to the creation of POSIX.2 as the first offshoot group in the POSIX family of standards. From this beginning, the POSIX.1 *sysconf()* function was extended and the *confstr()* function was created to provide an underlying implementation for the *getconf* utility, which allowed shell-level applications to query configuration-specific values such as maximum line length of text files. Once the beachhead of having functional interfaces in POSIX.2 was established, the temptation to continually add to this list has led to the current list as of Draft 9.

On the other hand, there are some very strong arguments against the inclusion of these functions. First, although the *regular expression* functions will almost certainly be required to implement many POSIX.2 utilities such as *ed*, *grep*, *awk*, *sed*, etc., these functions fall

short of the complete support needed to implement some utilities. For example, the handling of error messages (as in a syntactically incorrect regular expression) and the mechanisms of doing substitutions (including & and \n support) are not addressed. Because of this most implementors will be required to have "non-portable" proprietary extensions to their regular expression support to make a "commercially viable" implementation. The issue of where to draw the line between inclusion and exclusion is a difficult one indeed. Second, vendors and application writers may find it difficult, both procedurally and from a licensing perspective, to have part of the subroutine library come from a POSIX.1 developer and the other part implemented by the POSIX.2 implementor. For example, the implementor of *sysconf()*, *popen()*, or *system()* might do a much better job if common source code and assumptions were possible between the POSIX.1 and POSIX.2 APIs.

#### Status of POSIX.2 Balloting

"Dot 2 Classic" remains in its second round of balloting on Draft 9 with a new draft going to ballot in the June to July time frame, according to Hal Jespersen, the POSIX.2 Technical Editor.

During the Snowbird meeting, much of Monday was devoted to a presentation on the status of the Dot 2 Classic Balloting resolution. It is possible, and indeed likely, that Hal Jespersen will limit balloting on Draft 10 to unresolved objections and new material. If this is the case, it most likely indicates (although he didn't specifically say) that Hal has confidence that Draft 10 has a high probability of achieving the requisite 75% affirmative vote. Personally, I am not convinced that this is a likely event. While some decisions will be reversed (perhaps several times) before Draft 10, the following is a summary of issues and/or changes appearing in Draft 10:

- The internationalization utilities *locale* and particularly *localedef* are still controversial, particularly within AT&T. Because of the strong rationale for their existence it appears that they will remain in Draft 10, certainly with considerable amendment as the *UniForum Technical Committee on Internationalization* refines these newly developed utilities. This is just one case where the conflict between the role of standards to codify existing practice and the obvious holes in existing practice creates controversy. Perhaps this issue will be resolved by a balloting referendum such as was used for *uucp*.

- The Draft 10 shell will almost certainly strongly resemble that of Draft 9. Most of the important controversies appear to be largely settled and most changes appear to be corrections and clarifications.

- Most complex utilities, such as *awk*, *shell*, *lex*, *yacc*, etc., have undergone extensive reworking in response to ballot objections. Often a seemingly simple objection will cause large parts of the description to be rewritten in order to tighten it up with respect to completeness and clarity. I believe that Hal Jespersen believes that most of these changes are uncontroversial and he has ensured this by circulating draft sections via E-mail to various "experts." Certainly, many of these utilities desperately needed this clarification.

- It appears that the newly-engineered *hexdump* utility is to be replaced by a (much simpler) reversion to *od*. While *od* is the existing practice, the POSIX *od* will be a superset of the original one with most useful functionality in the new parts. It is not clear that hiding new invention under the same name is any less controversial than advertising its existence.

- Of course, there will be innumerable other changes, obviously important to many, that cannot for reasons of space be covered here.

A mock ballot on Draft 4 of the UPE was sent to the working group during February 1990 to allow ballot resolution to be the main focus of the Salt Lake meeting this April.

#### Status of the New Orleans Meeting

Monday, the working group reviewed the current status of the balloting on "Dot 2 Classic." This has already been discussed in earlier in this report.

The other four days were spent reviewing the 600 to 700 objections produced by the mock balloting process for the UPE. While the

number of objections seems low compared to the rate of objections for the corresponding number of pages in Dot 2 Classic, this may simply be a symptom of a general shortage of time and the lower number of people (generally 15 to 20) in the UPE working group. This lower number and general lack of time is a reflection of the fragmentation of the entire POSIX process caused by a proliferation of working groups.

Most of the work during mock balloting was of the nature of cleaning up incomplete or poorly worded textual descriptions. Particularly controversial issues were often left in the rationale for Draft 5. Some controversial utilities were moved to an appendix, based upon the belief that they should be removed while still allowing the balloting group one last chance to save them. The *lint89* was one such utility whose *raison d'etre* was meager. At best, the functionality probably should be an option to *c89* in the "Dot 2 Classic" document. The *sdiff* utility, which was inadvertently omitted from Draft 4, is to be included in Draft 5.

Altogether, it appears that Draft 5 is in a relatively healthy state to survive the rigors of the balloting process. Nonetheless, I expect that there will be a greater number of objections in the balloting this summer than there were in the mock ballot.

## Report on IEEE 1003.3: Test Methods

Doris Lebovits <lebovits@attunix.att.com> reports on the April 23-27 meeting in Salt Lake City, UT:

Dot three's job is to do test methods for all of the other 1003 standards. The group's work, whose first parts are now in ballot, specifies the requirements for OS conformance testing for our industry and for NIST. This makes what the balloting group and technical reviewers are doing, and their schedules, worth watching. Pay attention, also, to what comes out of the Steering Committee on Conformance Testing (SCCT). Their projects and decisions will be interesting and important.

This was the working group's sixteenth meeting. As usual, we reviewed the ballot status of P1003.1 test methods, worked on P1003.2 test methods, and reviewed steering committee activities. As before, each morning we did technical reviews of parts I and II; afternoons were spent writing assertions for part III. Participants from the usual companies attended (AT&T, NIST, OSF, Mindcraft, IBM, DEC, HP, Data General, Cray Research, Unisys, Perennial, and Unisoft Ltd.).

#### Document structure and some new PARs

Currently, our evolving document has two parts: Part I is generic test methods; Part II is test methods for measuring P1003.1 conformance, including test assertions; and Part III contains test methods and assertions for measuring P1003.2 conformance. (As other P1003 standards evolve, they will become separate activities in the working group's schedule.)

After the ballot, each part will become a separate standard. Part I will be published as IEEE P1003.3; Part II as IEEE P1003.3.1; and Part III as IEEE P1003.3.2. To this end, we developed and submitted three new PARs to the Standards Executive Committee (SEC). The PAR for P1003.3 lets Part I apply to all TCOS standards (i.e., POSIX). The PAR for P1003.3.1 lets Part II include test methods for P1003.1 and P1003.1a. The PAR for P1003.3.2 lets Part III include test methods for P1003.2.

#### Ballot status

Draft 11 of the current ballot, which was recirculated to the (approximately) ninety-member balloting group late in February, closed balloting March 23. Of the 65 respondents, 29 approved, 17 disapproved, and 19 abstained. This meets the two-thirds response requirement, but falls short of the needed two-thirds approval. Another re-circulation will probably take place in Fall, 1990.

#### P1003.2 verification

This was our fourth meeting working on a verification standard for the P1003.2 standard. The assertion writing and review were done in small groups. Some of the assertions were

based upon P1003.2 Draft 9. This group needs help from the P1003.2 working group in writing test assertions, but no formal arrangement is in place yet to provide it.

Officers for the P1003.2 Test Methods activities are: Ray Wilkes (Unisys), Chair; Lowell Johnson (Unisys) Secretary; and Andrew Twigger (Unisoft Ltd), Technical Editor.

#### Steering Committee on Conformance Testing (SCCT)

The test-methods steering committee is supposed to alleviate the increasing dot-three work load all the other proliferating groups are creating. Their job is coordinating the activities of all test-methods groups, monitoring their conformance to test methods, and writing Project Authorization Requests (PARs). Currently, its members are Roger Martin (NIST, Steering Committee Chair), Anita Mundkur (HP), Andrew Twigger (Unisoft Ltd), Bruce Weiner (Mindcraft), and Lowell Johnson (Unisys), but membership will be dynamic, Right now, this committee is documenting procedures. Roger Martin is also clarifying which standards the working group will address. The Technical Reviewers will review this work sometime before the next meeting.

## Report on IEEE 1003.4: Real-time Extensions

Rick Greer <rick@ism.isc.com> reports on the April 23-27 meeting in Salt Lake City, UT:

#### 1003.4

The .4 ballots went out on schedule, and most came back on schedule as well. We (barely) got the required 75% response, of which 43% approved of the draft as it stood. The small-group leaders are currently working to resolve the objections and will report back at Danvers in July.

#### 1003.4a

Most of the work at Snowbird centered around threads (.4a). Two alternatives to the pthreads proposal were presented at the meeting: "strands," from John Zolnowsky of Sun, defined a minimal set of interfaces for multi-threaded applications; "VP," from Paul Borman of Cray, added a "virtual processor" layer to the pthreads specification, which made some multiprocessor (MP) features visible to applications.

The primary MP hardware feature that Paul's VP proposal makes visible to the pthreads environment is the ability to write your own spin loops and expect them to work. One could, for example, have one thread continuously reading an in-core data base while another thread updates it. On an MP system, it might be more efficient to code this without using a mutex, although doing so on a uniprocessor with a co-routine threads package would be an absolute disaster. The new multiprocessor group, 1003.16, is looking into this and similar problems, and will probably suggest that .4a include some sort of system-wide attribute structure that one can check when writing programs that depend heavily on concurrent execution of threads.

After a week's discussion (often a euphemism for argument), we settled into a compromise position not that far from where we started–pthreads. All this work without much net change was frustrating, but probably unavoidable. Until fairly recently most of the committee was busy getting the .4 draft ready for balloting. Lacking enough time to have studied threads carefully, members were unwilling to accept the small group's conclusions before investigating some alternatives for themselves. Still, some progress was made. The most important was a more comprehensive definition of signal behavior in multi-threaded programs.

#### 1003.14

On the last day, a first attempt at a real-time application environment profile (AEP) was presented. This PAR will be an excellent, practical test of AEPs. Real-time applications are likely to vary wildly in the subsets of .4's rich features that they require. Some worry that the real-time AEP will force embedded systems that need only one or two .4 features to incorporate others just to adhere to the standard. The problem this poses is not just storage space wasted by unused code, but the expense of verifying that this extra code will

never get in the way of the application. The group will be wrestling with these and similar problems in the months to come.

## Report on IEEE 1003.4: Real-time Extensions

John Gertwagen <jag@ism.isc.com> reports on the April 23-27 meeting in Salt Lake City, UT:

### Administrivia

P1003 met in Salt Lake City this time. Actually, it was at Snowbird Lodge, south of and well above the city. It was spring in Salt Lake but still winter in the mountains. (Wish I skied.) The real-time meetings drew 68 people the first day, and averaged around 40 all week. If some skiers hadn't deserted each day, we would have had more.

### .4 Balloting

Over 200 people joined the balloting group for P1003.4, Draft 9. The first ballot closed in mid-March and 75% of balloters returned their ballots within a day or two of the official deadline, setting a new record! 43% of these voted "Yes" on the first round, about average for POSIX ballots.

Lack of time and logistics problems meant little ballot feedback by meeting-time (*shame* on those who didn't submit their ballots electronically!), but a few issues surfaced. Several objected to having binary semaphores only in the path namespace and not also in shared memory, where they could use simple test-and-set calls, and not time-consuming system calls. There's value in providing a common interface for both of these and for other "synchronization objects."

There were also objections to having "events" when there are "fixed" signals in System V, Release 4. The technical reviewer for events will try to make SVR4 signals meet real-time requirements. (Not too long ago, there were strong objections to changing signals. There may still be protests over adding real-time-required determinism.)

### Current Work

With .4 in limbo, the working group got on with Threads (.4a), Language Independent Bindings (.4b), and Real-time Application Environment Profiles (.14). Threads got the most attention. (Surprised?) Despite this, or perhaps because of it, the other two drafts saw significant progress.

Rick Greer has reviewed a lot of the threads work, so I'll briefly mention what's going on in .4b and .14, give you some personal views on threads, and then amplify on areas where our editor, Jeff Haemer, was recently raked over the coals.

### AEP

At first, the real-time AEP small group had some trouble focusing. They've identified two fairly easy targets, essentially minimum and maximum configurations, and now seek proposals for intermediate specifications.

In Utah, the group came up with a fairly complete specification for embedded systems, and reviewed it with P1003.0 *(EM the POSIX Guide group that is the driving force in defining AEPs. One interesting issue surfaced during the review: for embedded systems, the AEP group wants to *exclude* interfaces of .4 and .1 that aren't needed! Dot zero hadn't thought of that before. Resolving this should set an interesting precedent.

### Language-Independent Bindings

The people doing this have it down to a science, so the large group has largely left them alone. Most of their work is converting things to "normal" form, which is mostly tabular, and throwing away the stuff that is language-dependent. They made good use of their time, cranking through a good bit of the .4 draft.

### Threads (P1003.4a)

The meeting saw two new proposals. Both suggested fruitful changes to the current Pthreads work, but neither was accepted as a new base for the current draft.

John Zolnowsky of Sun Microsystems submitted one counter-proposal, called "strands" because "threads" was already taken. It was

an attempt to limit the scope of the interfaces and keep thread semantics closer to process semantics. Thus, it would have done away with mutexes and conditions, leaving synchronization to be accomplished through .4 binary semaphores, presumably modified to have inter-thread, not just inter-process, semantics. It also proposed more process-like exit semantics and a version of per-thread signaling. The consensus on the strands proposal seems to have been that it was too minimal.

In contrast, the VP (Virtual Processor) proposal, by Paul Borman of Cray Research, proposed significant "incremental" functionality in the form of a lower-level virtual-processor interface for use by the multi-processing and parallel processing communities. (For those familiar with Mach, it is roughly to Pthreads as cthreads is to Cthreads.) Features of the VP proposal included:

- *fork()* and *exec()* semantics for VPs
- per-VP signal semantics
- locks and events for synchronization
- no ordering or scheduling constraints

The group had several concerns about VP0

- Could it support real-time requirements without ordering and scheduling constraints?
- Could the Pthreads constraints be implemented on top of a layer that didn't support them?
- Would the interfaces be used by applications or by system implementors?
- Would an application using both Pthreads and VP interfaces encounter analogous problems to those caused by *read()*s and *fread()*s on the same file?
- Would standard interfaces for locks and events, implemented in hardware on many systems, constrain or encourage hardware development?
- Would the standard benefit either the user or vendor community?
- How soon could the proposal be completed and gain enough of the MP community's consensus to go to ballot?

Perhaps the deciding factor, though, was that the multi-processing AEP group (P1003.16) started meeting officially at Snowbird. [Editor: Watch for the snitch report, coming soon.] A majority of our group (including me) felt that MP-specific standards should grow from requirements identified by .16, not be created on the fly by the real-time working group.

In areas that are still not pinned down, the group made progress towards a better-defined cancellation mechanism, towards a "signals compromise" that improves on one hurriedly forged at the previous meeting, and towards more process-like exit semantics. The consensus was that we should try to accommodate and specify per-thread signal state. Although there are a few strong supporters, a majority did not feel that specification of per-thread signals is essential to a standard.

Paul Borman of Cray Research will present a proposal on this at the next meeting. I'll be interested to see what Paul comes up with. With three state elements (mask, pending signals, and action vector) and at least three signal delivery types (one, some, all), I can create many implementation models and corresponding application architectures. It may prove easy to construct a plausible model, but hard to construct one that 40 engineers can agree to live with for a long time! I suspect a portable application can assume nothing more than "exactly one signal gets delivered exactly once to exactly one handler." (Looks an awful lot likes signals to a process, doesn't it?)

The biggest progress in the meetings was wide consensus achieved for the current threads proposal. The working group resolved many of the remaining threads issues, and we let Bill Corwin tell IEEE/ISO that we expect to ballot P1003.4a in July, after the next meeting.

**OSF and UI Cooperating?**

Our editor's recent editorial stirred up a hornet's nest. (It wasn't so much what Jeff said as what he implied.) In his follow-up posting, he said I'd speak about the joint meetings in more detail. I didn't really want to but he twisted my arm, so here goes.

The UI MP Working Group and OSF have been cooperating since the middle of last year. I happen to work for a company that belongs to both, INTERACTIVE Systems Corporation, and though I haven't been to all of the joint meetings, I've attended them off and on since last November (which is, I think, when they started). Those I have attended focused on finding solutions to interface/semantic problems that both OSF and UI can endorse and that P1003.4 would probably endorse as well. Although these meetings haven't been advertised I've seen at least one article about OSF/UI/ATT negotiations that mentioned their cooperation in the MP arena. And the meetings have been open. At least one non-member has shown up uninvited, and was not asked to leave.

Now, it's no secret that several Pthreads proposal initiators (instigators?) work for OSF sponsors. Since the Pthreads-proposal was advanced before OSF adopted Mach, it's hard to say whether OSF influenced the P1003.4 work or the other way around. Also, in several instances, OSF/UI members have voted personal opinions contrary to the OSF/UI consensus established at the joint meetings.

What's the point? The joint meetings contribute to the quality of the .4a work, but they don't drive it. I think the work in P1003.4 is pushing vendors to find multi-threading solutions faster than they would have on their own. It's an example of POSIX pushing emerging technologies, not just creating standards. There's even a chance that .4a will create a standard multi-threading interface before millions of installed heterogeneous systems force the standard to a lowest common denominator or to incorporate a particular implementation's garbage.

And POSIX is playing another role – uniting the industry. I believe Sun's tooth-and-nail fights with AT&T in P1003.1 led to their current cooperation. Maybe the collaboration of OSF and UI on threads will also bring more unity to the industry.

**The relationship between .4 and .4a**

Despite what some think, the threads small group has never had any official status. Interest and participation in the threads effort goes far beyond the small group, or even the .4 working group into other POSIX committees. Some history may clear this up.

Lightweight processes (i.e., threads) have been on P1003.4's list of potential work items since its formation. About 3 years ago, the working group voted not to pursue them because they were not clearly needed and the technology was not sufficiently mature.

About a year and a half ago, threads resurfaced as an item of interest to the real-time users, and also to Ada, Transaction Processing, and RPC working groups. A small band of "experts" went off to draft a proposal. Since P1003.4 was an active system-interfaces committee and the real-time user community wanted a threads proposal, a lot of hard work culminated last summer in Minneapolis in a threads proposal being accepted as an additional chapter for the .4 draft.

There are 12 other interface proposals in the .4 draft. Some have been mature for nearly two years, (some with broad consensus, others without), others are still relatively wet behind the ears. Still, all the interfaces are relatively complete (sometimes too complete?), and in November, when it seemed appropriate to send .4 to ballot, .4a wasn't as complete as the rest. At the Milpitas meeting, the P1003.4 working group decided to include the threads chapter in the ballot for comment only, and sought and obtained authorization to turn the threads work into a separate work item for the P1003.4 working group.

After the Pthreads proposal was accepted into .4, the small group of people whose primary interest was threads spent all their time on threads. Meanwhile many other .4 members time-shared all the other .4 activities. Because the Pthreads supporters were so focused, they sometimes seemed like a separate group. (Some in the small group might have been surprised to learn they weren't. It takes a while to understand the POSIX bureaucracy.) Nevertheless, though they may not always have appeared to represent the

entire working group, the Pthreads proposal now enjoys wide consensus. Apparently, the small group has listened well to the interests of the working group and the POSIX community.

At Snowbird, there wasn't a threads small group, there were seven of them! These small groups examined how the current and the alternative proposals addressed:

- thread management
- synchronization
- signals/asynch events
- cancellation
- thread-specific data
- re-entrant functions
- process control

After reviewing all the issues, we discovered a consensus in most of these areas, and fairly strong agreement on most issues in the three or four groups that are still needed. It looks like things are pretty well on target.

I'm partially responsible for pushing .4a in before .4 was done, so I'm also partially responsible for the process not always appearing fair or well organized. I'll take my share of the blame. But I'll also take my share of the credit for progress in a technology that I believe to be important for real-time and for the entire POSIX community.

## Report on IEEE 1003.5: Ada bindings

Jayne Baker <cgb@d74sun.mitre.org> reports on the April 23-27 meeting in Salt Lake City, UT:

### Overview

The Utah meeting was the group's first since our October meeting in Brussels. In the interim, we had completed a mock ballot of Draft 4.0. Jim Lonjers of Unisys, one of our two co-chairs, managed the effort. The document was mailed out to reviewers on December 1, 1989 and comments were due January 19, 1990. Although only 16% of the ballots were returned, the high quality of the comments received made the mock ballot a success. Ted Baker, of Florida State University, hosted a special meeting in Tallahassee, March 19-23, to resolve issues and comments; the result was draft 4.1. We did not attend the January New Orleans meeting because balloters lacked sufficient time to review and return comments prior to the meeting, though some members came to attend other groups' meetings.

Our main goal in Utah was to prepare the Ada Language Binding Document for IEEE and ISO Ballot. We addressed the few unresolved technical issues from mock ballot; read draft 4.1 cover to cover, for accuracy (of text and Ada code), content, and consistency; established a plan for addressing the ISO formatting issues; adopted an optimistic schedule for IEEE and ISO ballots; and tried to establish a position on threads.

### Unresolved Technical Issues from Mock Ballot

Most unresolved technical issues from the mock ballot were trivial, and quickly resolved. They included the details of iterations (e.g., through a directory), string lower bounds with respect to a string returned by a function, the behavior of a file that is opened non-blocking when the I/O operation cannot complete, static initialization versus "easy implementation" of constants, and Text I/O page terminators.

The most detailed discussion involved whether or not files should be closed on an *Exec*. The Ada binding provides a *Start_Process* function, which is a primitive that safely creates a new process. In the face of Ada tasking, *Fork* and *Exec* are unsafe and cannot be used to accomplish the results of a *Start_Process* call. Once one of these unsafe primitives is issued, an application program is no longer under the control of the Ada run time system; the operating system is involved. Therefore, the integrity of the child process is jeopardized, and the state of the process's I/O (i.e., which files are open and closed) is not guaranteed. Application programs that must be safe with Ada tasking and must have files closed and buffers flushed should call *Start_Process* to create a new process.

**Global Issues Affecting the Document**

We solved several global editorial issues. We agreed to use a terse wording style except where a more lengthy explanatory style is needed for clarity. We accepted the current packaging of the Ada code (multiple packages) and the non-Ada-Language-Reference-Manual coding style. Chapter authors were assigned action items to complete their respective references and rationale sections.

We spent a large portion of the meeting going through the document chapter-by-chapter, noting very specific changes. Changes recorded in a "master red-lined" copy were forwarded to appropriate chapter authors at the close of the meeting. These changes will be made before the June delivery of the document to WG 15.

**ISO Format Issues**

We need to make several minor modifications, additions, and deletions before the June WG 15 meeting, to put the document in ISO standard format. After the March Tallahassee meeting, Jim Moore, of IBM, investigated the possibility of hiring a consulting technical editor to do this work. IBM volunteered to fund this effort at a level sufficient to translate the document into ISO format, maintain that format, and make one major edit and two to three minor editorial revisions. We accepted IBM's offer, and hired Hal Jespersen.

**Threads Issues**

As in New Orleans, several group members met with P1003.4 for threads discussions. Most group members feel we should establish a position on threads, but we remain firmly divided on what it should be. Several members believe the currently defined primitives (i.e., the most basic system functions) are insufficient, and think that any thread model and primitives proposed should be sufficient to support Ada tasking, and implement an Ada Run-Time. In contrast, at least one group member believes we are unrealistic to require a threads proposal in C to meet Ada requirements – we should, instead, require that C and Ada be able to play together in some reasonable fashion, and have a fair understanding of how it will be accomplished. We decided to admit our dissension to P1003.4. Interested P1003.5 members are acting as liaisons to represent their own views, but these liaisons do not represent any consolidated P1003.5. view.

**The IEEE and ISO ballots**

Steve Deller, our chair, asked the Sponsor's Executive Committee (SEC) to approve our entry into the IEEE ballot process. Jim Isaak, SEC Chair, met with us early in the week to discuss the IEEE and ISO ballot processes and help us establish a schedule to reach IEEE and ISO ballots simultaneously. Since the ballot process seems to be of general interest, here is a brief overview.

A hierarchy of organizations is responsible for producing international operating system standards and managing the ISO ballot process. Two independent international standards organizations, the International Standards Organization (ISO) and the International Electrotechnical Committee (IEC), sit on top. Joint Technical Committee 1 (JTC 1), a combined effort of these two organizations designed to coordinate their efforts in areas of overlap, is at the second level; Subcommittee 22 (SC 22), Languages, at the third; and Working Group 15 (WG 15), Portable Operating Systems for Computer Environments, at the fourth. National organizations, such as the American National Standards Institute (ANSI), manage ISO balloting within each country. Each participating country has one or more representatives in WG 15. The United States has a Technical Advisory Group (TAG), which meets with and advises the United States' WG 15 representatives on the U.S.'s position on important issues.

This bureaucracy requires quite a bit of coordination and planning to coordinate IEEE and ISO ballots. Most documents require about one year to complete the IEEE ballot cycle. Historically, POSIX documents have begun with the IEEE ballot process; three to four months later, either the original draft, or a newer version incorporating IEEE ballot process comments, enters the ISO process, and is delivered to both WG 15 and SC 22 for approval. Typically, the IEEE ballot is held open

until all comments from both IEEE and ISO processes are received, reviewed, and incorporated. The result is returned to both the IEEE and ISO ballot groups for their approval. If the IEEE comments are substantive, they enter into the ISO process by the submission of a United States position. For example, P1003.1a is the U.S. position on P1003.1..

Our group also initiated formation of a formal ballot group – the group that will actually vote on the current draft. We will deliver Draft 5.0, in ISO format, to WG 15 at the Ada Europe meeting this June. Draft 6.0 will go to IEEE ballot on August 6. If we receive the required 75% response by September 21, the ballot will close immediately; if not, we must reconsider the ballot group membership, and revise our schedule. In early October, draft 6.0 will be delivered to SC22. At the October meeting, in Seattle, we will resolve the IEEE ballot comments and produce Draft 7.0, which will enter the ISO Ballot process. At the January 1991, New Orleans Meeting, we will determine whether a second IEEE Ballot is needed. Any changes to Draft 7.0 resulting from a second IEEE Ballot will be entered into the ISO process through a *pro forma* objection. There are no guarantees, but P1003.5 could reach Draft International Standard (DIS) status by late second quarter of 1991.

#### Conclusion

The April '90 Salt Lake City Meeting was a success. We addressed the issues we hoped to address and attained our goal for the meeting. We also established a schedule for reaching IEEE and ISO ballot; although this schedule is optimistic, we think we can meet it.

## Report on IEEE 1003.6: Security

An anonymous source reports on the April 23-27 meeting in Salt Lake City, UT:

#### Apologia

This is my first and last review as a snitch. [Editor: We thank you for doing it, and hope your circumstances change to allow you to file more.] In it, you'll see no party line. My views will sometimes be controversial, and I hope they spark discussion and feedback. They represent neither the views of my company nor of its clients – I'm submitting this anonymously so no one can misconstrue them as being my company's – and they're certainly not meant to represent the consensus of the 1003.6 Working Group.

I'll put my biases on the table. I'm a commercial user and commercial software provider, not a government user, government software provider, or UNIX vendor. To some degree, these biases have influenced the committee, since I've been active in the group since its inception and attended every 1003.6 meeting. With that perspective, let's begin.

#### Overview

The 1003.6 Working Group is putting together a Department-of-Defense-inspired version of UNIX. Our efforts will help vendors sell systems to the U.S. Government and its contractors. All our interfaces will make it easier to evaluate conforming systems at one of the DoD's Trusted Computer Security Evaluation Criteria (TCSEC) levels. This is not inherently bad, but it does sell the commercial and international communities short. (More on this later.)

The working group is considering four areas: Discretionary Access Control (DAC), Mandatory Access Control (MAC), Least Privilege, and Audit.

**Discretionary Access Control**: The DAC group's job is hard. They are devising an Access Control List (ACL) mechanism that must co-exist with the familiar user, group, other mechanism. ACLs are discretionary because the user, not the system, decides each object's access rights. The traditional user, group, other mechanism is also discretionary: file protections are specified by the user. ACLs extend this by allowing users to grant different access permissions to arbitrary lists of named users and groups. (In other words, the traditional mechanism is an ACL with exactly three entries.) Designing an ACL is easy; maintaining compatibility with *chmod*, *stat*, *umask*, and the file creation mask of *creat* isn't.

**Mandatory Access Control**: MAC is another type of access control mechanism. All system

objects get a security label and all system users have a security classification set by the system or the Security Administrator (Systems Administrator). Users have no control over this mechanism's application; objects created by a user of classification *X* automatically receive a security label of *X*. Only users with appropriate classifications can access or modify a system object. (As a useful, if inexact, analogy, think of the way UNIX automatically assigns file ownerships.)

The TCSEC security criteria's popularity and widespread acceptance have given MAC another connotation – that of a codification of the familiar U.S. government, hierarchical security classifications: *Top Secret*, *Classified*, and *Unclassified*. Government policy prohibits users of a lower classification from viewing work of a higher classification. Conversely, users at a high classification may not make their work available to users at a lower classification: one can neither "read up" nor "write down." There are also compartments within each classification level, such as *NATO, nuclear, DOE,* or *project X*. Access requires the proper level *and* authorization for all compartments associated with the resource. The MAC group is defining interfaces for such a mandatory mechanism. It's not as confusing as it sounds, but outside of the DoD it *is* as useless as it sounds. (Prove me wrong. Show me how this DoD policy is useful in a commercial environment.)

**Least Privilege**: The Least Privilege group is eliminating root. They're creating both a list of privileges to encompass all of root's special uses (e.g., set-uid to a different user-id, create a directory, create a file system, override DAC protection) and a mechanism to inherit, assign, and enable those privileges.

**Audit**: The Audit group is preparing a standard interface for a logging mechanism, a standard format for logging records, and a list of system calls and commands to log.

#### Standards

At the ISO level, there will be no separate security standard. Our work will be merged with the 1003.1 (System Interface), 1003.2 (Commands and Utilities), and 1003.7 (System Administration) work in the ISO 9945-1, -2, and -3 standards. This means every conforming system will include security mechanisms. I like this. Do you?

#### Scope and motivation

All 1003.6 members feel we are making POSIX secure, not merely helping sell systems to the U.S. government. Our work is important and necessary (except, of course, MAC), but I think our focus has been too narrow. We included mechanisms for the TCSEC criteria but stopped there. We haven't sought out the work of other countries. We haven't considered the work being done in international standards bodies such as ISO and CCITT. We haven't explicitly considered commercial users. We've limited ourselves to helping provide TCSEC-conforming systems. Many of us believe that the TCSEC criteria are good for commercial applications. Is that hopeful claim just self-serving? We don't know. I wish eminent computer scientists and researchers had gotten together to study the needs of commercial users and drawn up an independent set of commercial security requirements. But they didn't.

Kevin Murphy, of British Telecom, is the ISO/IEC JTC1/SC22/WG15 security rapporteur – he formally represents the international community's concerns and views. In January, Kevin brought several of these to the working group's attention, including our TCSEC biases and lack of attention to ISO activities. The international set seems to consider the document's constant references to the TCSEC work provincial and inconsiderate of other countries' requirements. They also feel we should be more aware and accepting of ISO terminology in the document. Kevin also says our scope is too limited in the CCITT X.400 and X.500 areas.

#### Snowbird

**Plenary**: The meeting opened with a short plenary session. This time, the first topic of discussion was the progress of the 1003.6 draft document. Mike Ressler, of Bellcore, accepted the position of technical editor and brought a new draft of 1003.6, which contained work of all but the Audit subgroup. In addition, an

electronic copy of the document was available for the subgroups to modify and update during the meeting. The technical editor position had been open since October. No draft was available during this time, which worried us since it prevented us from setting any realistic completion date. With a draft in hand and a technical editor we now project completion in April, 1991.

Charlie Testa's absence meant we lacked our usual detailed report on TRUSIX. (TRUSIX is a DoD-sponsored organization made up of the National Computer Security Center, AT&T, and several other companies.) Rick Sibenaler and Shaun Rovansek, of the NCSC, gave us a brief update, reporting that the audit rationale will be available at the July POSIX meeting and that select experts are now reviewing the draft version of their formal model, which is written in a formal verification language, INA JO.

Some of the work of TRUSIX augments the work of 1003.6 – pursuit of a formal security model and descriptive top-level specification, and a mapping between them, for example – but some overlaps. I'm still puzzled over why TRUSIX has pursued audit and DAC mechanisms when 1003.6 is doing the same work. (Another challenge: can anyone out there tell me?) To their credit, TRUSIX is accomplishing their goals much faster than 1003.6. For example, Charlie reported in January that the TRUSIX DAC work is already complete. This speed may be at the expense of POSIX, since many very good people in both organizations are forced to split time between the two unnecessarily.

Mike Ressler reported on the networking and administration and security liaison group, which spends an afternoon at every POSIX meeting discussing mutual concerns of these three independent working groups. Here are the liaison group's goals, in areas of our common interest:

- identify areas of overlapping or missing coverage,
- provide an interface to ISO, ECMA, CCITT, and other international bodies, and
- exchange ideas and discuss related issues.

Peter Cordsen, of DataCentralen (Denmark), presented Danish security requirements. They define three levels of sensitivity, with criminal data among the most sensitive. There was no specific comparison to either the U.S. TCSEC or the emerging European security criteria. Peter suggested that the security working group begin addressing authentication, a position that received much support from other representatives.

**Draft work**: After the plenary, we worked on the document in subgroups.

*Discretionary Access Control (DAC)*: The group put together a new outline for the general and introductory sections of the draft and rewrote those sections to follow the new outline. They also resolved several issues:

- There will be only one type of default ACL, not the previously planned separate types for regular files and directories.
- A *mask* entry type has been added to provide a mechanism that temporarily overrides all other entries without actually changing their values or deleting them from the ACL. The feature also fits nicely with the current plan for ACL interaction with the old POSIX permission bits.
- The user model for both default and actual ACLs will be the same. (The internal representations are undefined.) System interfaces will be the same, too. A flag will be added to any interfaces that need to be able to distinguish the two.

**Audit**: Olin Sibert, of Sun, presented a new compromise audit proposal, based on an earlier one by Kevin Brady, of AT&T, and Doug Steves, of IBM, which he thought resolved some of the earlier work's problems. The working group accepted Olin's proposal with minor changes and incorporated it into Draft 6, which was distributed in the IEEE May mailing.

**Mandatory Access Control (MAC)**: Since Kevin Brady, the MAC chair, was participating in the Audit discussion, and Chris Hughes, of ICL, the acting chair, was also absent, Joe Bulger, of

NCSC, ran the meeting. It is still unclear who will chair the MAC subgroup.

Through the joint efforts of Bellcore and AT&T, the MAC draft had been translated from a proprietary word-processor format into the [n|t]roff + POSIX-macro format required for inclusion in the draft standard. The MAC draft's contents had been stable for several meetings, so the group spent the entire week changing the document.

This group seems to be having the most difficulty getting its job done. There doesn't seem to be as much discussion and active participation in the MAC group as the others.

**Privileges**: No functional changes were made to the privileges material at this meeting, but significant changes were made to the rationale. The group also firmed up concepts and disambiguated functional ambiguities.

**Networking, Administration, and Security Liaison**: The networking, administration, and security liaison group held its second meeting Wednesday afternoon. The meeting, chaired by Mike Ressler, started by reviewing the group's scope and goals.

Since there had been no ISO meeting since the January POSIX meeting, Yvon Klein, of Group Bull (France), didn't have anything new to say about ISO's security activities.

As part of the group's continuing efforts to identify problem areas, the system administration group and two networking groups gave presentations on their work. Steve Carter, of Bellcore, presented the scope and charter of the system administration group, 1003.7, and explained their use of an object-oriented paradigm. Jim Oldroyd, of the Instruction Set, followed this by presenting the work of 1003.7's interoperability subgroup.

Kester Fong, of General Motors, gave an overview of the FTAM group. He left us with the impression that there wasn't much room for collaboration, but we'll surely need to review the relationship between the file-system's security semantics and those of FTAM.

Jason Zions, of HP, gave one of the most interesting and aggressive presentations of the day, on the work of the Transparent File Access Group, which included a preliminary list of issues that 1003.8 feels need to be reviewed.

Finally, David Rogers, of ICL (Britain), gave a presentation on the European security criteria. He predicted harmonization by June 1990 of the work of Britain, France, Germany, and Holland. The European criteria will define separate levels of functionality and assurance. There will be ten classes of functionality. The first five are hierarchical and are similar to the U.S. Orange-Book criteria; the remaining five address particular security needs, such as integrity, availability, and networks. There are seven classes of assurance. A product evaluated under these criteria is likely to receive a rating from the first five functional classes, one or more of the next five functional classes, and an assurance rating.

**Final Comments**: With the short plenary session, the availability of the draft document in electronic form, and the presence of many lap-top systems to work on, this meeting was one of our most productive. The group seems to have picked up enthusiasm from the knowledge that our work is coming together and the end is in sight.

## Report on IEEE 1003.7: System Administration, Interoperability Subgroup

Jim R. Oldroyd <jr@inset.com> reports on the April 23-27 meeting in Salt Lake City, UT:

POSIX has given P1003.7 a charter to define both command-line and applications-programming interfaces for administering multiple networked machines from a central point. Most reports on this group seem to focus on the group's object-oriented approach: the administerable classes the group is defining, their attributes (properties) and their operators. [Editor: Martin Kirk has promised us a report on this. Watch for it soon.]

Sometimes overlooked in this object-oriented frenzy is another, equally important,

and perhaps more difficult goal of the group: *interoperability*.

Imagine, for example, an administrator who wishes to execute an operation on some fraction of nodes in a large, heterogeneous network of POSIX systems. The administrator wants to be able to issue the request once – and at his or her own terminal. The system should take care of determining which actual objects are affected and of communicating the request to them.

How should this be done? The fact that today's networks are heterogeneous means that it is not sufficient for vendors simply to supply systems with a consistent set of administerable object classes. Nor is it enough for vendors to define a consistent set of commands and API names that operate on these classes. On top of this, there has to be a consistent language for systems from different vendors to communicate with each other in order to tell each other that changes have to be made to some of the objects they are supporting.

The P1003.7 Interoperability subgroup is defining a standard protocol for communication with remote objects.

Currently, we are trying to work out the protocol's requirements. The protocol will have to support varied system-management philosophies. Some operations, such as re-enabling all PostScript† printers, should be queued and executed independently for each target. Failure to enable one printer does not mean that the other printers should remain disabled. Others operations must be atomic over the domain, for example, when adding a user to a set of machines, it is necessary to confirm that a UID is available on all target machines before adding the user to any machine.

Each of these problems saddles the protocol with a different requirement. The former case could be handled by broadcasting an instruction and collecting success or failure reports later; the latter requires a two-phase commit, requesting confirmation that successful completion is possible throughout the domain before actually mandating the change.

Do we have to invent a new protocol from scratch? P1003.7 is actively studying existing protocols, such as ISO's CMIP/CMIS and the Internet SNMP. Both of these are existing protocols designed to manage objects across multiple systems – exactly as per P1003.7's needs. However, both of these are actually designed to manage the network itself, and it is not clear that they lend themselves to management of things like users, printers, and filesystems (etc.) properly. We hope to discover whether some existing protocol will fill the bill in the next few meetings.

The Interoperability subgroup of P1003.7 will continue work in this area at our next meeting (Danvers, MA, July 16-20). If you are an interested party, we want to hear from you.

## Report on IEEE 1003.9: FORTRAN bindings

Michael Hannah <mjhanna@sandia.gov> reports on the April 23-27 meeting in Salt Lake City, UT:

**FORTRAN bindings committee prepares to go to ballot**

The FORTRAN bindings committee is preparing the official call for a ballot group. Because the POSIX work is all done under the auspices of the IEEE Technical Committee on Operating Systems Standards Subcommittee (TCOS-SS), all members of the ballot group must be both regular IEEE or Computer Society members. and members of the TCOS-SS (no extra charge to join). Non-members may submit informative ballots, but such ballots cannot count towards the required response percentage (75%), or percentage of affirmative responses (also 75%) required for passage of the standard. [Editor: Institutional Representatives are exceptions to this rule. See IEEE 1003.1-1988, p. 177 for a detailed explanation of the rules.]

For more information, the appropriate membership forms, and instructions for

† PostScript is a trademark of Adobe Systems, Inc.

returning the forms to the proper IEEE offices, contact the committee chair, John McGrory, at the address listed at the end of this article. This information and sign-up packet will be available by the end of June, but you may contact the chair as soon as you want your name added to the distribution list.

The formal sign-up period is expected to be August 15 through October 19, 1990. The ballot period is expected to last from November 9, 1990 through January 4, 1991. We are especially eager to attract a large representative balloting group, and encourage interested individuals to sign up. While the views represented on the P1003.9 working group have been appropriate and varied, the number of active members has been small (typically, around a dozen).

**Some history**

As the committee prepares to go to ballot, it might be of value to review some of the more sticky issues that the working group has addressed. The formal adopted charter of the committee is to provide access to the POSIX-defined standard operating system interface and environment, directly from the FORTRAN language. There are two major issues of scope that bear comment: "Access to how much of POSIX?" and "Which FORTRAN?"

Some POSIX features are easily imagined as useful to a FORTRAN application (e.g., chmod, exec, etc.); some are less easily imagined (pick your favorite obscure system call). It was unclear where to draw the line, so the committee took the approach of ensuring access to all features defined in 1003.1 (IEEE 1003.1-1988, or ISO/IEC 9945-1:1990). It seemed clear that full functional access would be provided by most vendors, so full standardization seemed called for. Some diehard C language addicts continue to ask, "Why have any FORTRAN bindings?" Although most vendors provide a method of calling C functions from FORTRAN, they vary from vendor to vendor. Further, any library of C routines provided by a vendor to map FORTRAN constructs to the POSIX defined procedures is bound to differ among vendors. The P1003.9 bindings are silent on implementation, so the FORTRAN subprograms defined in the bindings could be implemented as just such a library. The bindings just standardize the interface. Keeping in mind the POSIX goal of application portability, only a truly complete FORTRAN binding would provide portability of any FORTRAN application.

A harder issue was, "Which FORTRAN?" Our choices were:

1. FORTRAN 77 [ANSI X3.9-1978, ISO 1539-1980 (E)],

2. a codification of common extensions & enhancements to FORTRAN 77, or

3. the revised FORTRAN standard emerging from the ANSI X3J3 committee *(EM previously referred to as FORTRAN 8X but now called Fortran 90. (The working group has been delighted to have an officially appointed representative of X3J3 as an active member.) [Editor: Note that Fortran 90 will finally let us type the name of the language without using the caps-lock key. "And gain is gain, however small." – Robert Browning]

We chose the first.

For FORTRAN 77 vs. Fortran 90, we were swayed by the fact that FORTRAN 77 is currently the only adopted standard. (Fortran 90 is scheduled to be adopted as an ANSI standard *after* P1003.9 goes to ballot.) Further, FORTRAN-77-based applications are expected to exist for some years. Thus, the working group felt that FORTRAN-77-based bindings would be of value to the user community. The working group expects to develop a new set of bindings, based solely on Fortran 90, after completion of the FORTRAN 77 bindings (and after the Fortran 90 standard is adopted). One result of this decision is a subprogram-naming scheme that reflects the version of the language (e.g., *CALL F77MKDIR(...)* ). This will ensure that there will be no name-space conflict with similar-purpose subprograms in a future Fortran 90 binding.

An even harder issue, once we decided to base the bindings on FORTRAN 77, was whether to define the bindings as extensions and/or enhancements to the language itself, or simply as a library of callable FORTRAN subprograms. While the latter was finally chosen, there was considerable argument for

the former. In fact, one extension to FORTRAN 77 was considered minimally essential. The current document requires the language to differentiate external names unique to 31 characters, even though the FORTRAN 77 standard limits them to six. The extension seems harmless. Fortran 90 specifies uniqueness to 31 characters and all current FORTRAN 77 compilers researched provide this extension. Further, since the list of P1003.9 subprogram names is finite, if necessary, a vendor could provide a preprocessor to convert these names into unique strings of six characters.

If the P1003.9 bindings had defined changes to the language itself, then major missing constructs in the FORTRAN 77 language needed for easy POSIX access (most notably, structures and pointers) could have been provided by choosing either the emerging Fortran 90 constructs or an existing vendor solution. At first the working group felt that this might be required for some access features. However, as we struggled with each issue, working papers and proposals were introduced that resolved every one with callable FORTRAN subprograms (though some might argue about elegance or ease of use). While we mostly steered clear of "ease-of-implementation" arguments, since we viewed the FORTRAN 77 bindings as an interim, we felt that vendors would be quicker to implement a library of subprograms than modifications to compilers.

A final hard question of standard scope concerned whether to restrict the standard to 1003.1, or expand it to general FORTRAN-application portability issues, both within and outside the POSIX arena. Both a lack of resources and a desire to provide timely bindings on the heels of 1003.1 made us decide to limit the scope to 1003.1 functionality.

As other base standards are produced (e.g., 1003.2, 1003.4, etc.), we expect to construct and ballot bindings for those standards. For example, we have worked with P1003.2 in defining a standardized command to invoke the FORTRAN compiler (after a number of iterations, now named *fort77*) which is part of their current draft. Actual P1003.9 bindings to 1003.2 might include definitions of additional utilities of use to FORTRAN applications not mentioned in the base 1003.2 standard (e.g., *f77split*, *f77lint*, etc.).

Another argument against adding features was that many, if not most, of the problems we saw in portability are solved by new constructs in Fortran 90. Many of us felt that as a standards group we should only provide a minimum set of features for "perhaps-soon-to-be-obsolete" FORTRAN 77, and thereby speed up the date for providing full bindings to the new Fortran 90, which provides more features for application portability.

**How to get involved**

If you have strong feelings about these issues, the most effective avenue to express them at this point is to join the balloting group being formed. Nevertheless, if you wish to discuss them before this you can also directly contact the chair (John McGrory) or me (vice-chair, Michael Hannah), or join the e-mail discussion group. Addresses follow:

P1003.9 Chair:
John McGrory
Hewlett Packard Co.
Division 2615
19046 Pruneridge Avenue
Cupertino, Ca 95014
mcgrory%hpda@HPLABS.HP.COM

P1003.9 ViceChair:
Michael Hannah
Sandia National Labs
Albuquerque, NM 87185
mjhanna@SANDIA.GOV

Un-moderated mailing list:
posix-fortran@SANDIA.GOV

To join the list, send request to:
posix-fortran-request@SANDIA.GOV

## Report on 1003.12: Protocol-Independent Interfaces

Andy Nicholson <droid@earth.cray.com> reports on the April 23-27 meeting in Salt Lake City, UT:

**Introduction**

For starters, we've had some significant turnover. [Editor: Including, you'll note, Steve Head, our former fine dot 12 snitch. Thanks for diving in, Andy.] We are now down to five participants who were present a year ago, are on our third AT&T representative, and an HP representative has permanently left the working group. Despite all this, the group is beginning to make rapid advances on both the Simplified (SNI) and Detailed (DNI) network interfaces. This meeting's progress is sketched below.

**Overview of the Work We're Doing**

The dot 12 committee is working on three projects: Simplified Network Interface (SNI), Detailed Network Interface (DNI), and Data Representation Interface (DRI). Work on DRI is being delayed until SNI and DNI are well along. DNI is a protocol-independent interface to network services that allows access to protocol-dependent features in a protocol-independent manner. DNI is meant to provide a complete interface to everything you might expect to be able to do with networking services. DNI is comparable to Berkeley Sockets or AT&T's TLI, and we plan that anything that can be accomplished with those interfaces will be subsumed by DNI.

The idea behind SNI is that many applications will not require "detailed" access to networking services. SNI gives a "stdio" type of interface for networking that combines common groupings of procedures, eliminates access protocol-dependent features, and is just plain easier to use. Applications that use SNI aren't necessarily simple, they just don't need DNI's detailed access to networking services.

**Simplified Network Interface**

We started the week discussing the SNI interface. Norm Smith, from Unisys, had intended to bring an alternate SNI proposal to this meeting, but his group at Unisys decided to work with the current one. Irene Hu, from DEC, says she may yet offer an alternate proposal.

I presented a paper, prepared from previous minutes, which gathered up some deferred issues relating to SNI and we resolved some of them. For example, we added some explicit goals for SNI that everyone seemed to have accepted implicitly, but were never made official.

We also considered creating a formal definition of SNI functionality, to help us determine whether any particular function should be included, but decided it would be more efficient to keep deliberating each case individually. We'll record the rationale for each as part of the standard document to help us avoid and respond to ballot objections.

- SNI functionality

A paper by Tim Kirby (who works with me at Cray Research) prompted the group to redefine a function call. *Sni_recv()*, was defined to discard excess data in a datagram when the buffer offered by an application isn't large enough. The new version of *Sni_recv()*, allows an application either to discard excess data or to perform multiple *sni_recv()* calls to read it all. It also allows applications to discard datagrams without reading them at all. Here, I think the group has noticeably extended the power of the interface without sacrificing efficiency.

- Kernel objects

Because SNI endpoints may not be kernel objects, we need to define semantics and interfaces that will allow SNI endpoints to survive *exec()*. Unfortunately, we disagree on the semantics of the endpoint-preservation procedures. Should multiple copies of the same endpoint exist in different processes' address spaces, as happens with *fork()* and *exec()*? Implementing the protocol stack in a user library creates multiple copies of state information for the same endpoint, and it may be impossible to keep them synchronized.

Some of us (Keith Sklower, from Berkeley, the author of the SNI document, and I) want to restrict endpoint semantics so that only one process may have a copy of an SNI endpoint; others (Irene and Norm) disagree and wish to allow multiple copies of SNI endpoints where the programmer wishes.

**Detailed Network Interface**

We discussed DNI procedures in detail for the first time and found tentative agreement on most of the many issues raised. Mike Karels, from Berkeley's Computer Science Research Group, presented an outline of required functionality. After discussing it, we agreed to make DNI endpoints POSIX file descriptors (as returned by *open()*) until we see a compelling counter-argument. I'll challenge you to offer one.

On Wednesday, Irene gave an overview of XTI. During the presentation, Torez Hiley, our new attendee from ATT, told us that XTI is being revised: input from vendors using the Berkeley socket interface is prompting the addition of many features. Torez will report on the upcoming revision at the July meeting. Where sockets and XTI/TLI differ, the best solution is not clear. Moreover, some features are absent or inadequately supported in both interfaces. Here, we have a lot of work to do and are just getting started. We're eager to hear whether the new XTI solves any of our problems.

## Report on IEEE 1003.14: Multiprocessing

Bill Cox <bill@attunix.att.com> reports on the April 23-27 meeting in Salt Lake City, UT:

The POSIX Multiprocessing Study Group had its first official meeting as P1003.14 in Utah, where the SEC approved its Project Authorization Request (PAR) for a Multiprocessing Execution Environment Profile. Multiprocessing systems have become cost-effective means for providing computing power, but with the advantages come some specific concerns that need to be addressed at the interface level. The goal of this work is to try to make POSIX safe for multiprocessing; a secondary goal is to try to make POSIX hospitable for multiprocessing. POSIX working groups do not necessarily share the concerns of the implementors and users of multiprocessing systems.

Bob Knighten (Encore) is the Chair, Bill Cox (AT&T UNIX Software Operation) is Secretary, and Dave Plauger (Alliant) is the Technical Editor. Officers must have a commitment of support from their employers, to ensure that they can attend working group meetings and devote necessary time to the purposes of the working group. 16 people attended the group meetings.

People interested in .4A (threads) have tended to be interested in .14 and vice versa. Many .14 members that have been meeting with P1003.4/.4A see substantial problems with pthreads in a multiprocessor environment, and I know at least eight people working on .4A that want to come work in .14.

The working group designated one official liaison to .4A, who was joined by two other tentative volunteers. We will also establish liaisons with .1, .6, .7, .8, .10, and .12.

During the week, we spent time in three areas.

1. We clarified the group's work items, and started work on the most important, the Application Environment Profile. (An AEP may specify relevant portions of other POSIX working groups' work, make choices where options are permitted, and specify behavior that a [draft] standard may have left undefined or unspecified.)

2. We discussed current conventional wisdom on multiprocessing. The discussions centered around presentations by BBN, Cray, Encore, AT&T USO, and Alliant on lessons they've learned.

3. We created two small groups.

The first began work on high-level requirements placed on pthreads by multiprocessing. Attendees included Rick Greer (Interactive), Mary Weeks (Sun), and Bill Cox (AT&T USO). Here are some requirements we feel strongly about:

- A library implementation of "user-level threads" is vitally important. User-level threads often must be multiplexed onto kernel-supported objects/processes/threads, largely for performance reasons. These kernel-supported objects, etc, are sometimes called "virtual processors," because they support an abstraction closer to that of a physical

processor, with interrupts/signals, and a significant amount of state. .4A should not deal with threads at this virtual processor level.

• The formal memory model of P1003.4/D9 section 13.1 must apply to .4A. This model defines the semantics of memory interaction that should be preserved in a multithreaded or multiprocessing environment.

• Global threads scheduling makes little sense in a multiprocessor, though such scheduling could be useful as a hint (like C's `register` declarations if you don't have enough registers). Global policy is difficult to implement in a multiplexed thread environment.

• use of attribute structures for mutual exclusion variables (in particular, for scheduling hints)

• Locks shouldn't be opaque, and programmers should be able to statically initialize them. The latter is important so that locks can be part of data structures, and not require time-consuming dynamic allocation and initialization.

• There must be only one set of libraries. There are performance reasons to have single-threaded libraries, i.e., libraries that are not thread-aware, for a uniprocessor or single-threaded applications. The group believes that the cost of maintaining such libraries is sufficiently high that a non-reentrant library or set of libraries should not be required.

The other group began work on the AEP itself.

Members of this small group, and their responsibilities, included

Dave Plauger (Alliant) — skeleton for the document,

Frank Lawlor (IBM) — checkpoint restart, review, and liaison with .10 and .7,

James Gibson (BBN) — review and liason with .2,

Bob Knighten (Encore) — review and liason with .4, and

Tom Weaver (IBM) — review and liason with .1 and .6.

This group identified several areas of concern:

• microtasking models

• checkpoint, snapshots, and core dump format/synchronization

• a general programming model

• dividing the "reading list" (other P1003 standards and drafts)

• determining focus (are we dealing with portability for application writers, users, and/or administrators?)

• standardizing system services

A sketch of the planned document includes:

• reference to TIMS

• multithreaded applications (.4A)

• HLL parallel applications (PCF FORTRAN, parallel C)

• IPC

## Report on ANSI X3B11.1: WORM File Systems

Andrew Hume <andrew@research.att.com> reports on the April 17-19 meeting in Tucson, AZ:

### Introduction

X3B11.1 is working on a standard for file interchange on write-once, non-sequential (random access) media: a portable file system for WORMs. This was our fourth meeting. With many major issues somewhat settled, our main remaining decisions are how to implement directory hierarchies and how to manage free space.

### Multi-Volume Sets

WORM file systems store a fair amount of information per file (such as most of the fields in *struct stat*), per directory, per partition, and per volume. A volume is a logical address

space associated with a piece of physical media. For example, a WORM disk that can only be accessed one side at a time would be two volumes. Each volume has a label block describing its size, partitions, directory hierarchies, free space management, and so on. (Free space management is discussed below; for now, this could mean a pointer to the next free block.)

Informally, *multi-volume sets* means files and directories can be spread over several volumes. Here are some requirements for this feature:

- A file can be bigger than a volume (file sizes are limited to 2**64 bytes).
- You can append to a file.
- You can update any part of a file.
- All volumes need not be simultaneously accessible. (That is, if you have a three volume set, you don't need three drives.)
- Each volume, and the whole volume set, must be consistent after an update.
- Usable, although perhaps not fast, on a single drive. The idea is that you can't mandate that the control structures for the volume set be distributed over the set, because that would mean that on single-drive systems, users might have to mount every volume to recover even a single file. We would like to require only that the user mount the volume the file is on and perhaps one other (the master volume).
- Each volume must be self-contained (for disaster recovery),
- Security control is per volume, directory, and file.

After convincing ourselves that we all spoke roughly the same language, we came to consensus decisions for the following questions:

- Can a file description point to extents (files are spread across a list of extents) on later volumes? (Yes)
- Can a directory entry point to a file description on a later volume? (Yes)
- Must a directory be completely contained on a single volume? (No) Why? There's no reason to require it. All implementations are likely to use the same primitives to manage files and directories (that is, they'll implement directories as files); if you can handle multi-volume files correctly, directories should be easy too. Some people were concerned about being able to get the directory in one (more or less) I/O or block (especially for MS-DOS) but that can't happen in general; directories and files are likely to be spread all over the volume.
- must all the file descriptions for a single directory hierarchy fit on a single volume? (no)
- should each volume of a volume set point to the volume describing the root of the main directory hierarchy for that set? (yes)

The above involve subtleties not readily apparent; details available on request.

**Directory Hierarchies**

Much discussion centered on how to implement directory hierarchies – at least, after our initial surprise discovery that we are committed to support multiple directory hierarchies. This commitment comes from the CD-ROM standard, ISO 9660, where the intent was to have an ASCII directory tree and one or more national-character-set trees.

[Editor: CD-ROMs, like WORMs, are on write-only media, but solve different problems. Both provide tremendous storage capacity, but CD-ROMs appear to the user to be read-only, while WORMs appear to provide read and write access. Nevertheless, on WORMs, writing to a file means either appending characters to a preallocated chunk of disk, or rewriting a new version of the file in a new place. Once a file, or file version, is discarded, the piece of the physical medium it's stored on is forever lost, not released for reuse. CD-ROMs are for storing the Encyclopedia Britannica; WORMs are for storing backups.]

Our basic choice is between what I call the scattered directory tree, which is much like the standard, UNIX file system, and path tables (linearized encodings of the tree structure). ISO 9660 supports both. Scattered directories are simpler to deal with and somewhat easier to update, but probably slow to access because

they require too much seeking. Path tables seem faster at first glance (large contiguous reads, etc.), but their simplicity and speed seem to evaporate when the tables are modified. There is no consensus on which method to use. I originally held out for two methods, a flexible one and a really fast one, but have come to the conclusion, reinforced by conversations with Ken Thompson, that it is better to have one flexible method in the standard – scattered directories – and handle accelerators, such as directory trees cached on magnetic disk, as system-dependent structures outside the standard.

Suppose you update a file; doesn't that mean you also have to rewrite the directory, and, therefore, its parent directory, and, therefore, its parent directory, and so on all the way up to the root directory? And the volume header? How do you *find* the root directory, the volume header, and so on? This stuff is not yet decided but we envision that the file description stuff will have preallocated spare space so that a few updates can be done without changing anything outside the file description. Once this space is full, the system will have to get free space elsewhere, which implies updating some other area describing the free space. The volume header is in a fixed location (probably 8KB in from the start of media) and will point to any later volume headers and other stuff (such as where the root of the various directory trees are).

Requirements for the directory hierarchy include space and time efficiency, robustness (e.g., to minimize damage from a single I/O error), a single fast structure (unlike ISO 9660's two), and that a directory entry for a file must be complete (that is, point to all the extents for that file).

#### Space Allocation and Management

We must support preallocation of space (e.g., "Reserve 40MB of contiguous space for file 'xyz'.") both for speed and to support systems like the MacIntosh. Because of the latter, we also need to support giving back unused reserved space for later use.

These two requirements appear to force the standard to address describing the free space in a volume set, which will also be important if the standard is extended to cover R/W optical disks, where freed blocks need to be cleared before reuse. The two choices appear to be run-length encodings of the free space or bitmap techniques. The former can degrade to being quite large, while the latter have a fixed, but high, overhead (current media hold up to 8.2GB/side!).

#### Finale

We hope to conduct a letter ballot soon after the October 1990 meeting. If we can approve a proposal by January 1991 then it may be an ANSI standard by January 1992. Our next meeting is in Murray Hill, New Jersey, on July 17-19, where we expect to adopt the proposal being edited by Howard Kaikow as our working proposal. Anyone interested in attending should contact either the chairman, Ed Beshore (edb@hpgrla.hp.com), or me (andrew@research.att.com).

While this standard may seem of limited interest, because it deals only with WORMs, X3B11.1 expects approval shortly to develop a similar standard for R/W optical media. It doesn't take much imagination to see that standard being extended to apply to all rewritable direct-access media. (Unlike the CD-ROM standards committee, which ignored UNIX, this committee has a significant number of UNIX users, including representatives from AT&T Bell Labs, Sun, Hewlett-Packard. That, at least, ensures filenames won't be required to have a compulsory three-character suffix and a version number.) Once we have a working paper, anyone who cares about portable, multi-volume, multiple-character-set file systems should take a look. [Editor: Pay attention. He's giving you fair warning.]

## Report on Recent Standards Activities

Jeffrey S. Haemer <jsh@ico.isc.com> reports on spring-quarter standards activities:

This editorial is an overview of some of the spring quarter standards activities covered by the USENIX Standards Watchdog Committee. A companion article provides a general overview of the committee itself.

In this article, I've emphasized non-technical issues, which are unlikely to appear in official minutes and mailings of the standards committees. Previously published articles give more detailed, more technical views on most of these groups' activities. If my comments move you to read one of those earlier reports that you wouldn't have read otherwise, I've served my purpose. Of course, on reading that report you may discover the watchdog's opinion differs completely from mine.

**SEC: Standard/Sponsor Executive Committee**

The biggest hullabaloo in the POSIX world this quarter came out of the SEC, the group that approves creation of new committees. At the April meeting, in a move to slow the uncontrolled proliferation of POSIX standards, the institutional representatives (IRs) (one each from Usenix, UniForum, X/Open, OSF, and UI) recommended two changes in the Project Authorization Request (PAR) approval process: (1) firm criteria for PAR approval and group persistence and (2) a PAR approval group that had no working-group chairs or co-chairs. Dale Harris, of IBM Austin, presented the proposal and immediately took a lot of heat from the attendees, most of whom are working-group chairs and co-chairs. (Dale isn't an IR, but shared the concerns that motivated the recommendations and asked to make the presentation.)

The chair, Jim Isaak, created an ad hoc committee to talk over the proposal in a less emotional atmosphere. Consensus when the committee met was that the problem of proliferating PARs was real, and the only question was how to fix it. The group put together a formal set of criteria for PAR approval (which John Quarterman has posted to comp.std.unix), which seems to have satisfied everyone on the SEC, and passed without issue. The criteria seem to have teeth: at least one of the Project Authorization Requests presented later (1201.3, UIMS) flunked the criteria and was rejected. Two others (1201.1 and 1201.4 toolkits and Xlib) were deferred. I suspect (though doubt that any would admit it) that the proposals would have been submitted and passed in the absence of the criteria. In another related up-note, Tim Baker and Jim Isaak drafted a letter to one group (1224, X.400 API), warning them that they must either prove they're working or dissolve.

The second of the two suggestions, the creation of a PAR approval subcommittee, sank quietly. The issue will stay submerged so long as it looks like the SEC is actually using the approved criteria to fix the problem.

Shane McCarron's column in the July *Unix Review* covers this area in more detail.

**1003.0: POSIX Guide**

Those of you who have read my last two columns will know that I've taken the position that dot zero is valuable, even if it doesn't get a lot of measurable work done. This time, I have to say it looks like it's also making measurable progress, and may go to mock ballot by its target of fourth quarter of this year. To me, the most interesting dot-zero-related items this quarter are the growing prominence of profiles, and the mention of dot zero's work in the PAR approval criteria passed by the SEC.

Al Hankinson, the chair, tells me that he thinks dot zero's biggest contribution has been popularizing profiles –
basically, application-area-specific lists of pointers to other standards. This organizing principle has been adopted not only by the SEC (several of the POSIX groups are writing profiles), but by NIST (Al's from NIST) and ISO. I suspect a lot of other important organizations will fall in line here.

Nestled among the other criteria for PAR approval, is a requirement that PAR proposers write a sample description of their group for the POSIX guide. Someone questioned why proposers should have to do dot zero's job for them. The explanation comes in two pieces. First, dot zero doesn't have the resources to be an expert on everything, it has its hands full just trying to create an overall architecture. Second, the proposers aren't supplying what will ultimately go into the POSIX guide, they're supplying a sample. The act of drafting that sample will force each proposer to think hard about where the new group would fit in the grand scheme, right from the start.

This should help ensure that the guide's architecture really does reflect the rest of the POSIX effort, and will increase the interest of the other groups in the details of the guide.

**1003.1: System services interface**

Dot one, the only group that has completed a standard, is in the throes of completing a second. Not only has the IEEE updated the existing standard–the new version will be IEEE 1003.1-1990–ISO appears on the verge of approving the new version as IS 9945-1. The major sticking points currently seem limited to things like format and layout – important in the bureaucratic world of international standards, but inconsequential to the average user. Speaking of layout, one wonders whether the new edition and ISO versions will retain the yellow-green cover that has given the current document its common name–the ugly green book. (I've thought about soaking mine in Aqua Velva so it can smell like Green Chartreuse, too.)

The interesting issues in the group are raised by the dot-one-b work, which adds new functionality. (Read Paul Rabin's snitch report for the gory details.) The thorniest problem is the messaging work. *Messaging*, here, means a mechanism for access to external text and is unrelated to *msgget()*, *msgop()*, *msgctl()*, or any other message-passing schemes. The problem being addressed is how to move all printable strings out of our programs and into external "message" files so that we can change program output from, say, English to German by changing an environmental variable. Other dot-one-b topics, like symbolic links, are interesting, but less pervasive. This one will change the way you write any commercial product that outputs text – anything that has *printf()* statements.

The group is in a quandary. X/Open has a scheme that has gotten a little use. We're not talking three or four years of shake-out, here, but enough use to lay a claim to the "existing practice" label. On the other hand, it isn't a very pleasant scheme, and you'd have no problem coming up with alternative candidates. The UniForum Internationalization Technical Committee presented one at the April meeting. It's rumored that X/Open itself may replace its current scheme with another. So, what to do? Changing to a new scheme ignores existing internationalized applications and codifies an untried approach. Blessing the current X/Open scheme freezes evolution at this early stage and kills any motivation to develop an easy-to-use alternative. Not providing any standard makes internationalized applications (in a couple of years this will mean any non-throw-away program) nonportable, and requires that we continue to have to make heavy source-code modifications on every port – just what POSIX is supposed to help us get around.

To help you think about the problem, here's the way you'll have to write the "hello, world" koan using the X/OPEN interfaces:

```
#include <stdio.h>
#include <nl_types.h>
#include <locale.h>
main()
{
  nl_catd catd;

  (void)setlocale(LC_ALL, "");
  /* error checking omitted for brevity */
  catd = catopen("hello", 0);
  printf(catgets(catd, 1, 1, "hello, world\n"));
}
```

and using the alternative, proposed UniForum interfaces:

```
#include <stdio.h>
#include <locale.h>
main()
{
    (void)setlocale(LC_ALL, "");
    (void)textdomain("hello");
    printf(gettext("hello, world\n"));
}
```

I suppose if I had my druthers, I'd like to see a standard interface that goes even farther than the UniForum proposal: one that adds a default message catalogue/group (perhaps based on the name of the program) and a standard, *printf*-family messaging function to hide the explicit *gettext()* call, so the program could look like this:

```
#include <stdio.h>
#include <locale.h>
```

```
#define printf printmsg
main()
{
    /* inescapable, required by ANSI C */
    (void) setlocale(LC_ALL, "");
    printf("hello, world\n");
}
```

but that would still be untested innovation.

The weather conditions in Colorado have made this a bonus year for moths. Every morning, our bathroom has about forty moths in it. Stuck in our house, wanting desperately to get out, they fly toward the only light that they can see and beat themselves to death on the bathroom window. I don't know what to tell them, either.

**1003.2: Shell and utilities**

Someone surprised me at the April meeting by asserting that 1003.2 might be an important next target for the FORTRAN binding group. ("What does that mean?" I asked stupidly. "A standard for a FORTRAN-shell?") Perhaps you, like I, just think of dot two as language-independent utilities. Yes and no.

First, 1003.2 has over a dozen function calls (e.g., *getopt()*). I believe that most of these should be moved into 1003.1. *System()* and *popen()*, which assume a shell, might be exceptions, but having sections of standards documents point at things outside their scope is not without precedent. Section 8 of P1003.1-1988 is a section of C-language extensions, and P1003.5 will depend on the Ada standard. Why shouldn't an optional section of dot one depend on dot two? Perhaps ISO, already committed to regrouping and renumbering the standards, will fix this. Perhaps not. In the meantime, there are functions in dot two that need FORTRAN and Ada bindings.

Second, the current dot two standard specifies a C compiler. Dot nine has already helped dot two name the FORTRAN compiler, and may want to help dot two add a FORTRAN equivalent of *lint* (which I've heard called "flint"). Dot five may want to provide analogous sorts of help (though Ada compilers probably already subsume much of *lint*'s functionality).

Third, more subtle issues arise in providing a portable utilities environment for programmers in other languages. Numerical libraries, like IMSL, are often kept as single large source files with hundreds, or even thousands, of routines in a single *.f* file that compiles into a single *.o* file. Traditional FORTRAN environments provide tools that allow updating or extraction of single subroutines or functions from such objects, analogous to the way *ar* can add or replace single objects in libraries. Dot nine may want to provide such a facility in a FORTRAN binding to dot two.

Anyway, back to the working group. They're preparing to go to ballot on the UPE (1003.2a, User Portability Extensions). The mock ballot had pretty minimal return, with only ten balloters providing approximately 500 objections. Ten isn't very many, but mock ballot for dot two classic only had twenty-three. It seems that people won't vote until they're forced to.

The collection of utilities in 1003.2a is fairly reasonable, with only a few diversions from historic practice. A big exception is *ps(1)*, where historic practice is so heterogeneous that a complete redesign is possible. Unfortunately, no strong logical thread links the 1003.2a commands together, so read the ballot with an eye toward commands that should be added or discarded.

A few utilities have already disappeared since the last draft. *Pshar*, an implementation of *shar* with a lot of bells and whistles, is gone.

*compress/uncompress* poses an interesting problem. Though the utility is based on clear-cut existing practice, the existing implementation uses an algorithm that is copyrighted. Unless the author chooses to give the algorithm away (as Ritchie dedicated his set-uid patent to public use), the committee is faced with a hard choice:

- They can specify only the user interface. But the purpose of these utilities is to ease the cost of file interchange. What good are they without a standard data-interchange format?
- They can invent a new algorithm. Does it make sense to use something that isn't field

tested or consistent with the versions already out there? (One assumes that the existing version has real advantages, otherwise, why would so many people use a copyrighted version?)

Expect both the first real ballot of 1003.2a and recirculation of 1003.2 around July. Note that the recirculation will only let you object to items changed since the last draft, for all the usual bad reasons.

**1003.3: Test methods**

The first part of dot three's work is coming to real closure. The last ballot failed, but my guess is that one will pass soon, perhaps as soon as the end of the year, and we will have a standard for testing conformance to IEEE 1003.1-1988.

That isn't to say that all is rosy in dot-one testing. NIST's POSIX Conformance Test Suite (PCTS) still has plenty of problems: misinterpretations of dot one, simple timing test problems that cause tests to run well on 3b2's, but produce bad results on a 30 mips machine and even real bugs (attempts to read from a tty without first opening it). POSIX dot one is far more complex than anything for which standard test suites have been developed to date. The PCTS, with around 2600 tests and 150,000 lines of code, just reflects that complexity. An update will be sent to the National Technical Information Service (NTIS–also part of the Department of Commerce, but not to be confused with NIST) around the end of September which fixes all known problems, but with a suite this large, others are likely to surface later.

By the way, NIST's dot one suite is a driver based on the System V Verification Suite (SVVS), plus individual tests developed at NIST. Work has begun on a suite of tests for 1003.2, based, for convenience, on a suite done originally for IBM by Mindcraft. It isn't clear how quickly this work will go. (For example, the suite can't gel until dot two does.) For the dot one work, NIST made good use of Research Associates–people whose services were donated by their corporations during the test suite development. Corporations gain an opportunity to collaborate with NIST and inside knowledge of the test suite. I suspect Roger Martin may now be seeking Research Associates for dot two test suite development. If you're interested in doing this kind of work, want to spend some time working in the Washington, D.C. area, and think your company would sponsor you, his email address is *rmartin@swe.ncsl.nist.gov.*

By the way, there are a variety of organizational and numbering changes happening in dot three. See Doris Lebovits's snitch report for details.

The Steering Committee on Conformance Testing (SCCT) is the group to watch. Though they've evolved out of the dot three effort, they operate at the TCOS level, and are about to change the way POSIX standards look. In response to the ever-increasing burden placed on the testing committee, the SCCT is going to recommend that groups producing new standards include in those standards a list of test assertions to be used in testing them.

Groups that are almost done, like 1003.2, will be grandfathered in. But what should be done with a group like dot four–not far enough along that it has something likely to pass soon, but far enough to make the addition of major components to its ballot a real problem. Should this case be treated like language independence? If so, perhaps dot four will also be first in providing test assertions.

**1003.4: Real-time extensions**

The base dot-four document has gone to ballot, and the ensuing process looks like it may be pretty bloody. Fifty-seven percent of the group voted against the current version. (One member speculated privately that this meant forty-three percent of the balloting group didn't read it.) Twenty-two percent of the group (nearly half of those voting against) subscribed to all or part of a common reference ballot, which would require that entire chapters of the document be completely reworked, replaced, or discarded. Subscribers to this common reference ballot included employees of Unix International and the Open Software Foundation, of Carnegie-Mellon University and the University of California at Berkeley, and of Sun Microsystems and Hewlett-Packard. (USENIX did not ballot

similarly, but only because of lack of time.) Some of these organizations have never before agreed on the day of the week, let alone the semantics of system calls. But then, isn't bringing the industry together one goal of POSIX?

Still, the document has not been returned to the working group by the technical editors, so we can assume they feel hopeful about resolving all the objections. Some of this hope may come from the miracle of formality. I've heard that over half of the common reference ballot could be declared non-responsive, which means that there's no obligation to address over half the concerns.

The threads work appears to enjoy a more positive consensus. At least two interesting alternatives to the current proposal surfaced at the April meeting, but following a lot of discussion, the existing proposal stood largely unchanged. I predict that the threads work, which will go to ballot after the base dot four document, will be approved before it. John Gertwagen, dot four snitch and chair of UniForum's real-time technical committee, has bet me a beer that I'm wrong.

**1003.5: Ada bindings &**
**1003.9: FORTRAN-77 bindings**

These groups are coming to the same place at the same time. Both are going to ballot and seem likely to pass quickly. In each case, the major focus is shifting from technical issues to the standards process and its rules: forming balloting groups, relations with ISO, future directions, and so on.

Here's your chance to do a good deed without much work. Stop reading, call someone you know who would be interested in these standards, and give them the name of someone on the committee who can put them into the balloting group. (If nothing else, point them at our snitches for this quarter: Jayne Baker, *cgb@d74sun.mitre.org*, for dot five, and Michael Hannah, *mjhanna@sandia.gov*, for dot nine.) They'll get both a chance to see the standard that's about to land on top of their work and a chance to object to anything that's slipped into the standard that doesn't make sense. The more the merrier on this one, and they don't have to go to any committee meetings. I've already called a couple of friends of mine at FORTRAN-oriented companies; both were pleased to hear about 1003.9, and eager to read and comment on the proposed standard.

Next up for both groups, after these standards pass, is negotiating the IEEE standard through the shoals of ISO, both getting and staying in sync with the various versions and updates of the base standard (1003.1a, 1003.1b, and 9945-1), and language bindings to other standards, like 1003.2 and 1003.4. (See my earlier discussion of dot two.) Notice that they also have the burden of tracking their own language standards. At least in the case of 1003.9, this probably means eventually having to think about a binding to X3J3 (Fortran 90).

**1003.6: Security**

This group has filled the long-vacant post of technical editor, and so is finally back in the standards business. In any organization whose ultimate product is to be a document, the technical editor is a key person. [We pause here to allow readers to make some obligatory cheap shot about editors.] This is certainly the case in the POSIX groups, where the technical editors sometimes actually write large fractions of the final document, albeit under the direction of the working group.

I'm about to post the dot six snitch report, and don't want to give any of it away, but will note that it's strongly opinionated and challenges readers to find any non-DoD use for Mandatory Access Control, one of the half-dozen areas that they're standardizing.

**1003.7: System administration**

This group has to solve two problems at different levels at the same time. On the one hand, it's creating an object-oriented definition of system administration. This high-level approach encapsulates the detailed implementation of objects interesting to the system administrator (user, file system, etc.), so that everyone can see them in the same way on a heterogeneous environment. On the other hand, the protocol for sending messages to these objects must be specified in detail. If it

isn't, manufacturers won't be able to create interoperable systems.

The group as a whole continues to get complaints about its doing research-by-committee. It's not even pretending to standardize existing practice. I have mixed feelings about this, but am unreservedly nervous that some of the solutions being contemplated aren't even UNIX-like. For example, the group has tentatively proposed the unusual syntax *object action.* Command names will be names of objects, and the things to be done to them will be arguments. This bothers me (and others) for two reasons. First, this confuses syntax with semantics. You can have the message name first and still be object-oriented; look at C++. Second, it reverses the traditional, UNIX verb-noun arrangement: *mount filesystem* becomes *filesystem mount.* This flies in the face of the few existing practices everyone agrees on. I worry that these problems, and the resulting inconsistencies between system administration commands and other utilities, will confuse users. I have a recurring nightmare of a long line of new employees outside my door, all come to complain that I've forgotten to mark one of my device objects, */dev/null*, executable.

With no existing practice to provide a reality check, the group faces an uphill struggle. If you're an object-oriented maven with a yen to do something useful, take a look at what this group is doing, then implement some of it and see if it makes sense. Look at it this way: by the time the standard becomes reality, you'll have a product, ready to ship.

**1003.10: Supercomputing**

This group is working on things many of us us old-timers thought we had seen the last of: batch processing and checkpointing. The supercomputing community, condemned forever to live on the edge of what computers can accomplish, is forced into the same approaches we used back when computer cycles were harder to come by than programmer cycles, and machines were less reliable than software.

Supercomputers run programs that can't be run on less powerful computers because of their massive resource requirements (cpu/memory/io). They need batch processing and checkpointing because many of them are so resource-intensive that they even run for a long time on supercomputers. Nevertheless, the supercomputing community is not the only group that would benefit from standardization in these areas. (See, for example, my comments on dot fourteen.) Even people who have (or wish to have) long-running jobs on workstations, share some of the same needs for batch processing and checkpointing.

Karen Sheaffer, the chair of dot ten, had no trouble quickly recasting the group's proposal for a batch PAR into a proposal that passed the SEC's PAR approval criteria. The group is modeling a batch proposal after existing practice, and things seem to be going smoothly.

Checkpointing, on the other hand, isn't faring as well. People who program supercomputers need to have a way to snapshot jobs in a way that lets them restart the jobs at that point later. Think, for example, of a job that needs to run for longer than a machine's mean-time-to-failure. Or a job that runs for just a little longer than your grant money lasts. There are existing proprietary schemes in the supercomputing world, but none that's portable. The consensus is that a portable mechanism would be useful and that support for checkpointing should be added to the dot one standard. The group brought a proposal to dot one b, but it was rejected for reasons detailed in Paul Rabin's dot one report. Indeed, the last I heard, dot-one folks were suggesting that dot ten propose interfaces that would be called from within the program to be checkpointed. While this may seem to the dot-one folks like the most practical approach, it seems to me to be searching under the lamp-post for your keys because that's where the light's brightest. Users need to be able to point to a job that's run longer than anticipated and say, "Checkpoint this, please." Requiring source-code modification to accomplish this is not only unrealistic, it's un-UNIX-like. (A helpful person looking over my shoulder has just pointed out that the lawyers have declared "UNIX" an adjective, and I should say something like "un-UNIX-system-like" instead. He is, of course, correct.)

Whatever the interface is, it simply must provide a way to let a user point at another process and say, "Snapshot it," just as we can stop a running job with job control.

#### 1003.12: Protocol-independent interfaces

This group is still working on two separate interfaces to the network: Simplified Network Interface (SNI) and Detailed Network Interface (DNI). The January meeting raised the possibility that the group would coalesce these into a single scheme, but that scheme seems not to have materialized. DNI will provide a familiar socket- or XTI/TLI-like interface to networks, while SNI will provide a simpler, stdio-like interface for programs that don't need the level of control that DNI will provide. The challenge of SNI is to make something that's simple but not so crippled that it's useless. The challenge of DNI is to negotiate the fine line between the two competing existing practices. The group has already decided not to use either sockets or XTI, and is looking at requirements for the replacement. Our snitch, Andy Nicholson, challenged readers to find a reason not to make DNI endpoints POSIX file descriptors, but has seen no takers.

#### 1003.14: Multiprocessing

The multiprocessing group, which had been meeting as sort of an ad hoc spinoff of the real-time group, was given PAR approval at the April meeting as 1003.16 but quickly renamed 1003.14 for administrative reasons. They're currently going through the standard set of jobs that new groups have to accomplish, including figuring out what tasks need to be accomplished, whom to delegate them to, and how to attract enough working-group members to get everything done. If you want to get in on the ground floor of the multiprocessing standard, come to Danvers and volunteer to do something.

One thing that needs to be done is liaison work with other committees, many of which are attacking problems that bear on multiprocessors as well. One example is dot ten's checkpointing work, which I talked about earlier. Checkpointing is both of direct interest to dot fourteen, and is analogous to several other problems the group would like to address. (A side effect of the PAR proliferation problem mentioned earlier is that intergroup coordination efforts go up as the square of the number of groups.)

#### 1201: Windows, sort of

Okay, as a review, we went into the Utah meeting with one official group, 1201, and four unofficial groups preparing PARs:

1. 1201.1: Application toolkit
2. 1201.2: Recommended Practice for Driveability/User Portability
3. 1201.3: User Interface Management Systems
4. 1201.4: Xlib

By the end of the week, one PAR had been shot down (1201.3), one approved (1201.2), and two remained unsubmitted.

The 1201.4 par was deferred because the X consortium says Xlib is about to change enough that we don't want to standardize the existing version. I'll ask, "If it's still changing this fast, do we want to even standardize on the next version?" The 1201.1 PAR was deferred because the group hasn't agreed on what it wants to do. At the beginning of the week, the two major camps (OSF/Motif and OPEN LOOK)† had agreed to try to merge the two interfaces. By mid-week, they wouldn't even sit at the same table. That they'd struck off in an alternative compromise direction by the end of the week speaks extremely highly of all involved. What the group's looking at now is a toolkit at the level of XVT:‡ a layer over all of the current competing technologies that would provide portability without invalidating any existing applications. This seems like just the right approach. (I have to say this because I suggested it in an editorial about six months ago.)

The 1201.3 PAR was rejected. Actually, 1201 as a whole voted not to submit it, but the people working on it felt strongly enough that

† OSF/Motif is a Registered Trademark of the Open Software Foundation.
OPEN LOOK is a Registered Trademark of AT&T.

‡ XVT is a trademark of XVT Software Inc.

they submitted it anyway. The SEC's consensus was that the field wasn't mature enough to warrant even a recommended practice, but the work should continue, perhaps as a UniForum Technical Committee. The study group countered that it was important to set a standard before there were competing technologies, and that none of the attendees sponsoring companies would be willing to foot the bill for their work within anything but a standards body. The arguments weren't persuasive.

The 1201.2 PAR, in contrast, sailed through. What's interesting about this work is that it won't be an API standard. A fair fraction of the committee members are human-factors people, and the person presenting the PAR convinced the SEC that there is now enough consensus in this area that a standard is appropriate. I'm willing to believe this, but I think that stretching the net of the IEEE's Technical Committee on Operating Systems so wide that it takes in a human-factors standard for windowing systems is overreaching.

X3

There are other ANSI-accredited standards-sponsoring bodies in the U.S. besides the IEEE. The best known in our field is the Computer Business Equipment Manufacturers' Association (CBEMA), which sponsors the X3 efforts, recently including X3J11, the ANSI-C standards committee. X3J11's job has wound down; Doug Gwyn tells me that there's so little happening of general interest that it isn't worth a report. Still, there's plenty going on in the X3 world. One example is X3B11, which is developing a standard for file systems on optical disks. Though this seems specialized, Andrew Hume suggests in his report that this work may eventually evolve into a standards effort for file systems on any read-write mass storage device. See the dot-four common reference ballot for the kind of feelings new file-system standards bring out.

I encourage anyone out there on an X3 committee who thinks the committee could use more user exposure and input to file a report. For example, Doug Gwyn suggests that there is enough activity in the C++ standards world to merit a look. If anyone out there wants to volunteer a report, I'd love to see it.

# Plan 9 from Bell Labs

*Rob Pike, Dave Presotto, Ken Thompson, Howard Trickey*
*rob@research.att.com*

*Bell Labs*

Plan 9 is a distributed computing environment. It is assembled from separate machines acting as CPU servers, file servers, and terminals. The pieces are connected by a single file-oriented protocol and local name space operations. By building the system from distinct, specialised components rather than from similar general-purpose components, Plan 9 achieves levels of efficiency, security, simplicity, and reliability seldom realised in other distributed systems. This paper discusses the building blocks, interconnections, and conventions of Plan 9.

## Introduction[1]

Plan 9 is a general-purpose, multi-user, portable distributed system implemented on a variety of computers and networks. It lacks a number of features often found in other distributed systems, including

i. A uniform distributed name space,

ii. Process migration,

iii. Lightweight processes,

iv. Distributed file caching,

v. Personalised workstations,

vi. Support for X windows.

Unhappy with the trends in commercial systems, we began a few years ago to design a system that could adapt well to changes in computing hardware. In particular, we wanted to build a system that could profit from continuing improvements in personal machines with bitmap graphics, in medium- and high-speed networks, and in high-performance microprocessors. A common approach is to connect a group of small personal timesharing systems – workstations – by a medium-speed network, but this has a number of failings. Because each workstation has private data, each must be administered separately; maintenance is difficult to centralise. The machines are replaced every couple of years to take advantage of technological improvements, rendering the hardware obsolete often before it has been paid for. Most telling, a workstation is a largely self-contained system, not specialised to any particular task, too slow and I/O-bound for fast compilation, too expensive to be used just to run a window system. For our purposes, primarily software development, it seemed that an approach based on distributed specialisation rather than compromise could better address issues of cost-effectiveness, maintenance, performance, reliability, and security. We decided to build a completely new system, including compiler, operating system, networking software, command interpreter, window system, and (on the hardware side) terminal. This construction would also offer an occasion to rethink, revisit, and perhaps even replace most of the utilities we had accumulated over the years.

1. This paper is reprinted with kind permission of the UKUUG and was delivered at the UKUUG conference in London in July 1990. See the end of this newsletter for abstracts of this conference.

Plan 9 is divided along lines of service function. CPU servers concentrate computing power into large (not overloaded) multiprocessors; file servers provide repositories for storage; and terminals give each user of the system a dedicated computer with bitmap screen and mouse on which to run a window system. The sharing of computing and file storage services provides a sense of community for a group of programmers, amortises costs, and centralises and hence simplifies management and administration.

The pieces communicate by a single protocol, built above a reliable data transport layer offered by an appropriate network, that defines each service as a rooted tree of files. Even for services not usually considered as files, the unified design permits some noteworthy and profitable simplification. Each process has a local file *name space* that contains attachments to all services the process is using and thereby to the files in those services. One of the most important jobs of a terminal is to support its user's customised view of the entire system as represented by the services visible in the name space.

To be used effectively, the system requires a CPU server, a file server, and a terminal; it is intended to provide service at the level of a departmental computer centre or larger. The CPU server and file server are large machines best housed in an air conditioned machine room with conditioned power. The system's strengths stem in part from economies of scale, and the scale we have in mind is large. One of our goals, perhaps unrealisable, is to unite the computing environment for all of AT&T Bell Laboratories (about 30,000 people) into a single Plan 9 system comprising thousands of CPU and file servers spread throughout, and clustered in, the company's various departments. That is clearly beyond the administrative capacity of workstations on Ethernets.

The following sections describe the basic components of Plan 9, explain the name space and how it is used, and offer some examples of unusual services that illustrate how the ideas of Plan 9 can be applied to a variety of problems.

## CPU Servers

Several computers provide CPU service for Plan 9. The production CPU server is a Silicon Graphics Power Series machine with four 25MHz MIPS processors, 128 megabytes of memory, no disk, and a 20 megabyte-per-second back-to-back DMA connection to the file server. It also has Datakit and Ethernet controllers to connect to terminals and non-Plan 9 systems [Fra80a, Met80a]. The operating system provides a conventional view of processes, based on `fork` and `exec` system calls [Ker84a], and of files, mostly determined by the remote file server. Once a connection to the CPU server is established, the user may begin typing commands to a command interpreter in a conventional looking environment [Duf90a, Rit74a].

A multiprocessor CPU server has several advantages. The most important is its ability to absorb load. If the machine is not saturated (which can be economically feasible for a multiprocessor) there is usually a free processor ready to run a new process. This is similar to the notion of free disk blocks in which to store new files on a file system. The comparison extends farther: just as one might buy a new disk when a file system gets full, one may add processors to a multiprocessor when the system gets busy, without needing to replace or duplicate the entire system. Of course, one may also add new CPU servers and share the file servers.

The CPU server performs compilation, text processing, and other applications. It has no local storage; all the permanent files it accesses are provided by remote servers. Transient parts of the name space, such as the collected images of active processes [Kil84a] or services provided by user processes, may reside locally but these disappear when the CPU server is rebooted. Plan 9 CPU servers are as interchangeable for their task – computation – as are ordinary terminals for theirs.

## File Servers

The Plan 9 file servers hold all permanent files. The current server is another Silicon Graphics computer with two processors, 64 megabytes of memory, 600 megabytes of magnetic disk, and a 300 gigabyte jukebox of write-once optical disk (WORM). (This machine is to be replaced by a MIPS 6280, a single processor with much greater I/O bandwidth.) It connects to Plan 9 CPU servers through 20 megabyte-per-second DMA links, and to terminals and other machines through conventional networks.

The file server presents to its clients a file system rather than, say, an array of disks or blocks or files. The files are named by slash-separated components that label branches of a tree, and may

be addressed for I/O at the byte level. The location of a file in the server is invisible to the client. The true file system resides on the WORM, and is accessed through a two-level cache of magnetic disk and RAM. The contents of recently-used files reside in RAM and are sent to the CPU server rapidly by DMA over a high-speed link, which is much faster than regular disk although not as fast as local memory. The magnetic disk acts as a cache for the WORM and simultaneously as a backup medium for the RAM. With the high-speed links, it is unnecessary for clients to cache data; instead the file server centralises the caching for all its clients, avoiding the problems of distributed caches.

The file server actually presents several file systems. One, the "main" system, is used as the file system for most clients. Other systems provide less generally-used data for private applications. One service is unusual: the backup system. Once a day, the file server freezes activity on the main file system and flushes the data in that system to the WORM. Normal file service continues unaffected, but changes to files are applied to a fresh hierarchy, fabricated on demand, using a copy-on-write scheme [Qui90a]. Thus, the file tree is split into two: a read-only version representing the system at the time of the dump, and an ordinary system that continues to provide normal service. The roots of these old file trees are available as directories in a file system that may be accessed exactly as any other (read-only) system. For example, the file `/usr/rob/doc/plan9.ms` as it existed on April 1, 1990, can be accessed through the backup file system by the name `/1990/0401/usr/rob/doc/plan9.ms`. This scheme permits recovery or comparison of lost files by traditional commands such as file copy and comparison routines rather than by special utilities in a backup subsystem. Moreover, the backup system is provided by the same file server and the same mechanism as the original files so permissions in the backup system are identical to those in the main system; one cannot use the backup data to subvert security.

## Terminals

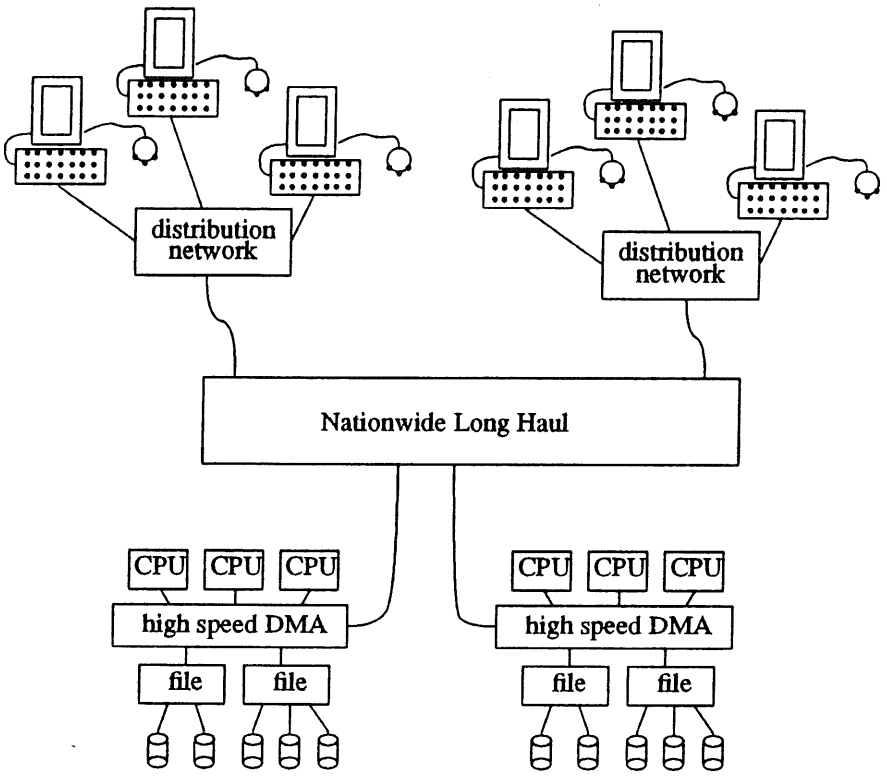
The standard terminal for Plan 9 is a Gnot (with silent "G"), a locally-designed machine of which several hundred have been manufactured. The terminal's hardware is reminiscent of a diskless workstation: 4 or 8 megabytes of memory, a 25MHz 68020 processor, a 1024×1024 pixel display with two bits per pixel, a keyboard, and a mouse. It has no external storage and no expansion bus; it is a terminal, not a workstation. A 2 megabit per second packet-switched distribution network connects the terminals to the CPU and file servers. Although the bandwidth is low for applications such as compilation, it is more than adequate for the terminal's intended purpose: to provide a window system, that is, a multiplexed interface to the rest of Plan 9.

Unlike a workstation, the Gnot does not handle compilation; that is done by the CPU server. The terminal runs a version of the CPU server's operating system, configured for a single, smaller processor with support for bitmap graphics, and uses that to run programs such as a window system and a text editor. Files are provided by the standard file server over the terminal's network connection.

Just like old character terminals, all Gnots are equivalent, as they have no private storage either locally or on the file server. They are inexpensive enough that every member of our research centre can have two: one at work and one at home. A person working on a Gnot at home sees exactly the same system as at work, as all the files and computing resources remain at work where they can be shared and maintained effectively.

## Networks

Plan 9 has a variety of networks that connect the components. CPU servers and file servers communicate over back-to-back DMA controllers. That is only practical for the scale of, say, a computer centre or departmental computing resource. More distant machines are connected by traditional networks such as Ethernet or Datakit. A terminal or CPU server may use a remote file server completely transparently except for performance considerations. As our Datakit network spans the country, Plan 9 systems could be assembled on a large scale, although this has not been tried in practice. (See Figure 1.)




To keep their cost down, Gnots employ an inexpensive network that uses standard telephone wire and a single-chip interface. (The throughput is respectable, about 120 kilobytes per second.)

To get even that bandwidth to home is of course problematic. Some of us have DS-1 lines at 1.54 megabits per second; others are experimenting with more modest communications equipment. Since the terminal only mediates communication – it instructs the CPU server to connect to the file server but does not participate in the resulting communication – the relatively low bandwidth to the terminal does not affect the overall performance of the system.

## Name Spaces

There are two kinds of name space in Plan 9: the global space of the names of the various servers on the network and the local space of files and servers visible to a process. Names of machines and services connected to Datakit are hierarchical, for example `nj/mh/astro/helix`, defining (roughly) the area, building, department, and machine in a department [Fra80a]. Because the network provides naming for its machines, global naming issues need not be handled directly by Plan 9. However one of Plan 9's fundamental operations is to attach network services to the local name space on a per-process basis. This fine-grained control of the local name space is used to address issues of customisability, transparency, and heterogeneity.

The protocol for communicating with Plan 9 services is file-oriented; all services must implement a file system. That is, each service, local or remote, is arranged into a set of file-like objects collected into a hierarchy called the name space of the server. For a file server, this is a trivial requirement. Other services must sometimes be more imaginative. For instance, a printing service might be implemented as a directory in which processes create files to be printed. Other examples are described in the following sections; for the moment, consider just a set of ordinary file servers distributed around the network.

When a program calls a Plan 9 service (using mechanisms inherent in the network and outside Plan 9 itself) the program is connected to the root of the name space of the service. Using the protocol, usually as mediated by the local operating system into a set of file-oriented system

calls, the program accesses the service by opening, creating, removing, reading, and writing files in the name space.

From the set of services available on the network, a user of Plan 9 selects those desired: a file server where personal files reside, perhaps other file servers where data is kept, or a departmental file server where the software for a group project is being written. The name spaces of these various services are collected and joined to the user's own private name space by a fundamental Plan 9 operator, called *attach,* that joins a service's name space to a user's. The user's name space is formed by the union of the spaces of the services being used. The local name space is assembled by the local operating system for each user, typically by the terminal. The name space is modifiable on a per-process level, although in practice the name space is assembled at log-in time and shared by all that user's processes.

To log in to the system, a user sits at a terminal and instructs it which file server to connect to. The terminal calls the server, authenticates the user (see below), and loads the operating system from the server. It then reads a file, called the *profile,* in the user's personal directory. The profile contains commands that define what services are to be used by default and where in the local name space they are to be attached. For example, the main file server to be used is attached to the root of the local name space, `/`, and the process file system is attached to the directory `/proc`. The profile then typically starts the window system.

Within each window in the window system runs a command interpreter that may be used to execute commands locally, using file names interpreted in the name space assembled by the profile. For computation-intensive applications such as compilation, the user runs a command `cpu` that selects (automatically or by name) a CPU server to run commands. After typing `cpu`, the user sees a regular prompt from the command interpreter. But that command interpreter is running on the CPU server *in the same name space – even the same current directory – as the* `cpu` *command itself.* The terminal exports a description of the name space to the CPU server, which then assembles an identical name space, so the customised view of the system assembled by the terminal is the same as that seen on the CPU server. (A description of the name space is used rather than the name space itself so the CPU server may use high-speed links when possible rather than requiring intervention by the terminal.) The `cpu` command affects only the performance of subsequent commands; it has nothing to do with the services available or how they are accessed.

Although there is a large catalogue of services available in Plan 9, including the service that finds services, a few suffice to illustrate the usage and possibilities of this design.

## The Process File System

An example of a local service is the "process file system", which permits examination and debugging of executing processes through a file-oriented interface. It is related to Killian's process file system [Kil84a] but its differences exemplify the way that Plan 9 services are constructed.

The root of the process file system is conventionally attached to the directory `/proc`. (Convention is important in Plan 9; although the name space may be assembled willy-nilly, many programs have conventional names built in that require the name space to have a certain form. It doesn't matter which server the program `/bin/rc` (the command interpreter) comes from but it must have that name to be accessible by the commands that call on it.) After attachment, the directory `/proc` itself contains one subdirectory for each local process in the system, with name equal to the numerical unique identifier of that process. (Processes running on the remote CPU server may also be made visible; this will be discussed below.) Each subdirectory contains a set of files that implement the view of that process. For example, `/proc/77/mem` contains an image of the virtual memory of process number 77. That file is closely related to the files in Killian's process file system, but unlike Killian's, Plan 9's `/proc` implements other functions through other files rather than through peculiar operations applied to a single file. Here is a list of the files provided for each process.

`mem`
: The virtual memory of the process image. Offsets in the file correspond to virtual addresses in the process.

`ctl`
: Control behaviour of the processes. Messages sent (by a `write` system call) to this file

cause the process to stop, terminate, resume execution, etc.

`text`
The file from which the program originated. This is typically used by a debugger to examine the symbol table of the target process, but is in all respects except name the original file; thus one may type "`/proc/77/text`" to the command interpreter to instantiate the program afresh.

`note`
Any process with suitable permissions may write the `note` file of another process to send it an asynchronous message for interprocess communication. The system also uses this file to send (poisoned) messages when a process misbehaves, for example divides by zero.

`status`
A fixed-format ASCII representation of the status of the process. It includes the name of the file the process was executed from, the CPU time it has consumed, its current state, etc.

The `status` file illustrates how heterogeneity and portability can be handled by a file server model for system functions. The command `cat /proc/*/status` presents (readably but somewhat clumsily) the status of all processes in the system; in fact the process status command `ps` is just a reformatting of the ASCII text so gathered. The source for `ps` is a page long and is completely portable across machines. Even when `/proc` contains files for processes on several heterogeneous machines, the same implementation works.

Whether the functions provided by the `ctl` file should instead be accessed through further files – `stop`, `terminate`, etc. – is a matter of taste. We chose to fold all the true control operations into the `ctl` file and provide the more data-intensive functions through separate files.

It is worth noting that the services `/proc` provides, although varied, do not strain the notion of a process as a file. For example, it is not possible to terminate a process by attempting to remove its process file nor is it possible to start a new process by creating a process file. The files give an active view of the processes, but they do not literally represent them. This distinction is important when designing services as file systems.

## The Window System

In Plan 9, user programs, as well as specialised stand-alone servers, may provide file service. The window system is an example of such a program; one of Plan 9's most unusual aspects is that the window system is implemented as a user-level file server.

The window system is a server that presents a file `/dev/cons`, similar to the `/dev/tty` or `CON:` of other systems, to the client processes running in its windows. Because it controls all I/O activities on that file, it can arrange that each window's group of processes sees a private `/dev/cons`. When a new window is made, the window system allocates a new `/dev/cons` file, puts it in a new name space (otherwise the same as its own) for the new client, and begins a client process in that window. That process connects the standard input and output channels to `/dev/cons` using the normal file opening system call and executes a command interpreter. When the command interpreter prints a prompt, it will therefore be written to `/dev/cons` and appear in the appropriate window.

It is instructive to compare this structure to other operating systems. Most operating systems provide a file like `/dev/cons` that is an alias for the terminal connected to a process. A process that opens the special file accesses the terminal it is running on without knowing the terminal's precise name. Since the alias is usually provided by special arrangement in the operating system, it can be difficult for a window system to guarantee that its client processes can access their window through this file. This issue is handled easily in Plan 9 by inverting the problem. A set of processes in a window shares a name space and in particular `/dev/cons`, so by multiplexing `/dev/cons` and forcing all textual input and output to go through that file the window system can simulate the expected properties of the file.

The window system serves several files, all conventionally attached to the directory of I/O devices, `/dev`. These include `cons`, the port for ASCII I/O; `mouse`, a file that reports the position of the mouse; and `bitblt`, which may be written messages to execute bitmap graphics primitives. Much as the different `cons` files keep separate clients' output in separate windows, the `mouse` and `bitblt` files are implemented by the window system in a way that keeps the various clients independent. For example, when a

client process in a window writes a message (to the `bitblt` file) to clear the screen, the window system clears only that window. All graphics sent to partially or totally obscured windows is maintained as a bitmap layer, in memory private to the window system [Pik83a]. The clients are oblivious of one another.

Since the window system is implemented entirely at user level with file and name space operations, it can be run recursively: it may be a client of itself. The window system functions by opening the files `/dev/cons`, `/dev/bitblt`, etc., as provided by the operating system, and reproduces – multiplexes – their functionality among its clients. Therefore, if a fresh instantiation of the window system is run in a window, it will behave normally, multiplexing *its* `/dev/cons` and other files for *its* clients. This recursion can be used profitably to debug a new window system in a window or to multiplex the connection to a CPU server [Pik89a]. Since the window system has no bitmap graphics code – all its graphics operations are executed by writing standard messages to a file – the window system may be run on any machine that has `/dev/bitblt` in its name space, including the CPU server.

## CPU Command

The `cpu` command connects from a terminal to a CPU server using a full-duplex network connection and runs a setup process there. The terminal and CPU processes exchange information about the user and name space, and then the terminal-resident process becomes a user-level file server that makes the terminal's private files visible from the CPU server. (At the time of writing, the CPU server builds the name space by re-executing the user's profile; a version being designed will export the name space using a special terminal-resident server that can be queried to recover the terminal's name space.) The CPU process makes a few adjustments to the name space, such as making the file `/dev/cons` on the CPU server *be the same file as on the terminal*, perhaps making both the local and remote process file system visible in `/proc`, and begins a command interpreter. The command interpreter then reads commands from, and prints results on, its file `/dev/cons`, which is connected through the terminal process to the appropriate window (for example) on the terminal. Graphics programs such as bitmap editors also may be executed on the CPU server since their definition is entirely based on I/O to files "served" by the terminal for the CPU server. The connection to the CPU server and back again is utterly transparent.

This connection raises the issue of heterogeneity: the CPU server and the terminal may be, and in the current system are, different types of processors. There are two distinct problems: binary data and executable code. Binary data can be handled two ways: by making it not binary or by strictly defining the format of the data at the byte level. The former is exemplified by the `status` file in `/proc`, which enables programs to examine, transparently and portably, the status of remote processes. Another example is the file, provided by the terminal's operating system, `/dev/time`. This is a fixed-format ASCII representation of the number of seconds since the epoch that serves as a time base for `make` and other programs [Ker84a]. Processes on the CPU server get their time base from the terminal, thereby obviating problems of distributed clocks.

For files that are I/O intensive, such as `/dev/bitblt`, the overhead of an ASCII interface can be prohibitive. In Plan 9, such files therefore accept a binary format in which the byte order is predefined, and programs that access the files use portable libraries that make no assumptions about the order. Thus `/dev/bitblt` is usable from any machine, not just the terminal. This principle is used throughout Plan 9. For instance, the format of the compilers' object files and libraries is similarly defined, which means that object files are independent of the type of the CPU that compiled them.

Having different formats of executable binaries is a thornier problem, and Plan 9 solves it adequately if not gracefully. Directories of executable binaries are named appropriately: `/mips/bin`, `/68020/bin`, etc., and a program may ascertain, through a special server, what CPU type it is running on. A program, in particular the `cpu` command, may therefore attach the appropriate directory to the conventional name `/bin` so that when a program runs, say, `/bin/rc`, the appropriate file is found. Although this is a fairly clumsy solution, it works well in practice. The various object files and compilers use distinct formats and naming conventions, which makes cross-compilation painless, at least once automated by `make` or a similar program

[Ker84a].

## Security

Plan 9 does not address security issues directly, but some of its aspects are relevant to the topic. Breaking the file server away from the CPU server enhances the possibilities for security. As the file server is a separate machine that can only be accessed over the network by the standard protocol, and therefore can only serve files, it cannot run programs. Many security issues are resolved by the simple observation that the CPU server and file server communicate using a rigorously controlled interface through which it is impossible to gain special privileges.

Of course, certain administrative functions must be performed on the file server, but these are available only through a special command interface accessible only on the console and hence subject to physical security. Moreover, that interface is for administration only. For example, it permits making backups and creating and removing files, but it does not permit reading files or changing their permissions. *The contents of a file with read permission for only its owner will not be divulged by the file server to any other user, even the administrator.*

Of course, this begs the question of how a user proves who he or she is. At the moment, we use a simple authentication manager on the Datakit network itself, so that when a user logs in from a terminal, the network assures the authenticity of the maker of calls from the associated terminal. In order to remove the need for trust in our local network, we plan to replace the authentication manager by a Kerberos-like system [Mil87a].

## Discussion

A fairly complete version of Plan 9 was built in 1987 and 1988, and then its development was abandoned for a number of aesthetic and technical reasons. In May of 1989 work was begun on a completely new system, based on the SGI MIPS-based multiprocessors, using the first version as a bootstrap environment. By October, the CPU server could compile all its own software, using the first-draft file server. The SGI file server came on line in February 1990; the true operating system kernel at its core was taken from the CPU server's system, but the file server is otherwise a completely separate program (and computer). The CPU server's system was ported to the 68020 in 13 hours elapsed time on November 12-13, 1989. One portability bug was found; the fix affected two lines of code. At the time of writing (April 1990), work has just begun on the new window system; it should be running well before this paper appears (July 1990). (Until it is complete, we will continue to use the terminal software from the 1987-1988 implementation.)

All the authors use Plan 9 almost exclusively; only the lack of an electronic mail facility, which is being addressed, prevents us from moving over permanently. Plan 9 is up and running and comfortable to use, although it is certainly too early to pass final judgement.

The multiprocessor operating system for the MIPS-based CPU server has 454 lines of assembly language, more than half of which saves and restores registers on interrupts. The kernel proper contains 3647 lines of C plus 774 lines of header files, which includes all process control, virtual memory support, trap handling, and so on. There are 1020 lines of code to interface to the 29 system calls. Much of the functionality of the system is contained in the "drivers" that implement built-in servers such as `/proc`; these and the network software add another 9511 lines of code. Most of this code is identical on the 68020 version; for instance, all the code to implement processes, including the process switcher and the `fork` and `exec` system calls, is identical in the two versions; the peculiar properties of each processor are encapsulated in two five-line assembler routines. (The code for the respective MMU's is quite different, although the page fault handler is substantially the same.) It is only fair to admit, however, that the compilers for the two machines are closely related, and the operating system may depend on properties of the compiler in unknown ways.

The system is efficient. On the four-processor machine connected to the MIPS file server, the 45 source files of the operating system compile in about ten seconds of real time and load in another ten. (The loader runs single-threaded.) Partly due to the register-saving convention of the compiler, the null system call takes only 7 microseconds on the MIPS, about half of which is attributed to relatively slow memory on the multiprocessor. A process fork takes 700 microseconds irrespective of the process's size.

Plan 9 does not implement lightweight processes explicitly. We are uneasy about deciding where

on the continuum from fine-grained hardware-supported parallelism to the usual timesharing notion of a process we should provide support for user multiprocessing. Existing definitions of threads and lightweight processes seem arbitrary and raise more questions than they resolve [Acc86a]. We prefer to have a single kind of process and to permit multiple processes to share their address space. With the ability to share local memory and with efficient process creation and switching, both of which are in Plan 9, we can match the functionality of threads without taking a stand on how users should multiprocess.

Process migration is also deliberately absent from Plan 9. Although Plan 9 makes it easy to instantiate processes where they can most effectively run, it does nothing explicit to make this happen. The compiler, for instance, does not arrange that it run on the CPU server. We prefer to do coarse-grained allocation of computing resources simply by running each new command interpreter on a lightly-loaded CPU server. Reasonable management of computing resources renders process migration unnecessary.

Other aspects of the system lead to other efficiencies. A large single-threaded chess database problem runs about four times as fast on Plan 9 as on the same machine running commercial software because the remote cache on the file server is so large. In general, most file I/O is done by direct DMA from the file server's cache; the file server rarely needs to read from disk at all.

Much of Plan 9 is straightforward. The individual pieces that make it up are relatively ordinary; its unusual aspects are in how the pieces are put together. As a case in point, the recent interest in using X terminals connected to timeshared hosts might seem to be similar in spirit to how Plan 9 terminals are used, but that is a mistaken impression. The Gnot, although similar in hardware power to a typical X terminal, serves a much higher-level function in the computing environment. It is a fully programmable computer running a virtual memory operating system that maintains its user's view of the entire Plan 9 system. It offloads from the CPU server all the bookkeeping and I/O intensive chores that a window system must perform. It is not really a workstation either; for example one would rarely bother to compile on the Gnot, although one would certainly run a text editor there. Like the other pieces of Plan 9, the Gnot's strength derives from careful specialisation in concert with other specialised components.

## Acknowledgements

Many people helped build the system. We would like especially to thank Bart Locanthi, who built the Gnot and encouraged us to program it; Tom Duff, who wrote the command interpreter `rc`, Tom Killian and Ted Kowalski, who cheerfully endured early versions of the software; and Dennis Ritchie, who frequently provided us with much-needed wisdom.

## Authors' Note

Since this paper was first written almost all the 'to be done' work mentioned in the paper, in particular the window system, has been done.

## References

[Acc86a] M. J. Accetta, Robert Baron, William Bolosky, David Golub, Richard Rashid, Avadis Tevanian, and Michael Young, "Mach: A New Kernel Foundation for UNIX Development", *USENIX Conference Proceedings*, Atlanta, GA (July, 1986).

[Duf90a] A T. Duff, "Rc – A Shell for Plan 9 and UNIX", *UNIX Programmer's Manual, Tenth Edition*, Murray Hill, NJ, AT&T Bell Laboratories, (1990).

[Fra80a] A. G. Fraser, "Datakit – A Modular Network for Synchronous and Asynchronous Traffic", *Proc. Int. Conf. on Commun.*, Boston, MA, (June 1980).

[Ker84a] Brian W. Kernighan, and Rob Pike, *The UNIX Programming Environment*, Prentice-Hall, Englewood Cliffs, NJ, (1984).

[Kil84a] T. J. Killian, "Processes as Files", *USENIX Summer Conference Proceedings*, Salt Lake City, UT, USA, (June 1984).

[Met80a] R. M Metcalfe, and D. R. Boggs, *The Ethernet Local Network: Three Reports*, XEROX Palo Alto Research Center, (February 1980).

[Mil87a] A S. P. Miller, C. Neumann, J. I. Schiller, and J. H. Saltzer, *Kerberos Authentication and Authorization System*, MIT, (1987).

[Pik83a] R. Pike, "Graphics in Overlapping Bitmap Layers", *Transactions on Graphics* 2(2), pp. 135-160 (1983).

[Pik89a] R. Pike, "A Concurrent Window System", *Computing Systems* 2(2), pp. 133-153 (1989).

[Qui90a] S. Quinlan, "A Cached WORM File System", *Software – Practice and Experience*, p. To appear (1990).

[Rit74a] D. M. Ritchie, and K. Thompson, "The UNIX Time-Sharing System", *Comm. Assoc. Comp. Mach.* 17(7), (July 1974).

# Rc – A Shell for Plan 9 and UNIX Systems

*Tom Duff*
*td@research.att.com*

*AT&T Bell Laboratories*

*Rc* is a command interpreter for Plan 9. It also runs on a variety of traditional systems, including SunOS and the Tenth Edition. It provides similar facilities to Bourne's */bin/sh*, with some small additions and mostly less idiosyncratic syntax. This paper introduces *rc*'s highlights with numerous examples, and discusses its design and why it varies from Bourne's.

## Introduction[1]

Plan 9 needs a command-programming language. As porting the Bourne shell to an incompatible new environment seemed a daunting task, I chose to write a new command interpreter, called *rc* because it runs commands. Although tinkering with perfection is a dangerous business, I could hardly resist trying to "improve" on Bourne's design. Thus *rc* is similar in spirit but different in detail from Bourne's shell.

The bulk of this paper describes *rc*'s principal features with many small examples and a few larger ones. We close with a discussion of the principles guiding *rc*'s design and why it differs from Bourne's design. The descriptive sections include little discussion of the rationale for particular features, as individual details are hard to justify in isolation. The impatient reader may wish to skip to the discussion at the end before skimming the expository parts of the paper.

## Simple commands

For the simplest uses *rc* has syntax familiar to Bourne-shell users. Thus all of the following behave as expected:

```
date
con alice
who >user.names
who >>user.names
wc <file
echo [a-f]*.c
who | wc
who; date
cc *.c &
cyntax *.c && cc -g -o cmd *.c
rm -r junk || echo rm failed!
```

## Quotation

An argument that contains a space or one of *rc*'s other syntax characters must be enclosed in apostrophes (').

```
rm 'odd file name'
```

An apostrophe in a quoted argument must be doubled:

```
echo 'How''s your father?'
```

## Variables

*Rc* provides variables whose values are lists of arguments. Variables may be given values by typing, for example:

```
path=(. /bin /usr/bin)
user=td
tty=/dev/tty8
```

The parentheses indicate that the value assigned to `path` is a list of three strings. The variables

1. This paper is reprinted with kind permission of the UKUUG and was delivered at the UKUUG conference in London in July 1990. See the end of this newsletter for abstracts of this conference.

`user` and `tty` are assigned lists containing a single string.

The value of a variable can be substituted into a command by preceding its name with a `$`, like this:

```
echo $path
```

If `path` had been set as above, this would be equivalent to

```
echo . /bin /usr/bin
```

Variables may be subscripted by numbers or lists of numbers, like this:

```
echo $path(2)
echo $path(3 2 1)
```

These are equivalent to

```
echo /bin
echo /usr/bin /bin .
```

There can be no space separating the variable's name from the left parenthesis. Otherwise, the subscript would be considered a separate parenthesized list.

The number of strings in a variable can be determined by the `$#` operator. For example,

```
echo $#path
```

would print the number of entries in `$path`.

The following two assignments are subtly different:

```
empty=()
null=''
```

The first sets `empty` to a list containing no strings. The second sets `null` to a list containing a single string, but the string contains no characters.

Although these may seem like more or less the same thing (in Bourne's shell, they are indistinguishable), they behave differently in almost all circumstances. Among other things

```
echo $#empty
```

prints 0, whereas

```
echo $#null
```

prints 1.

All variables that have never been set have the value `()`.

## Arguments

When *rc* is reading its input from a file, the file has access to the arguments supplied on *rc*'s command line. The variable `$*` initially has the list of arguments assigned to it. The names `$1`, `$2`, etc. are synonyms for `$*(1)`, `$*(2)`, etc. In addition, `$0` is the name of the file from which *rc*'s input is being read.

## Concatenation

*Rc* has a string concatenation operator, the caret `^`, to build arguments out of pieces.

```
echo hully^gully
```

is exactly equivalent to

```
echo hullygully
```

Suppose variable `i` contains the name of a command. Then

```
cc -o $i $i^.c
```

might compile the command's source code, leaving the result in the appropriate file.

Concatenation distributes over lists. The following

```
echo (a b c)^(1 2 3)
src=(main subr io)
cc $src^.c
```

are equivalent to

```
echo a1 b2 c3
cc main.c subr.c io.c
```

In detail, the rule is: if both operands of `^` are lists of the same non-zero number of strings, they are concatenated pairwise. Otherwise, if one of the operands is a single string, it is concatenated with each member of the other operand in turn. Any other combination of operands is an error.

## Free carets

User demand has dictated that *rc* insert carets in certain places, to make the syntax look more like the Bourne shell. For example, this:

```
cc -$flags $stems.c
```

is equivalent to

```
cc -^$flags $stems^.c
```

In general, *rc* will insert `^` between two arguments that are not separated by white space. Specifically, whenever one of `$'` ` follows a

quoted or unquoted word, or an unquoted word follows a quoted word with no intervening blanks or tabs, a ^ is inserted between the two. If an unquoted word immediately following a $ contains a character other than an alphanumeric, underscore or *, a ^ is inserted before the first such character.

## Command substitution

It is often useful to build an argument list from the output of a command. *Rc* allows a command, enclosed in braces and preceded by a left quote, `` `{...} ``, anywhere that an argument is required. The command is executed and its standard output captured. The characters stored in the variable `ifs` are used to split the output into arguments. For example,

```
cat `{ls -tr|sed 10q}
```

will catenate the ten oldest files in the current directory in temporal order.

## Pipeline branching

The normal pipeline notation is general enough for almost all cases. Very occasionally it is useful to have pipelines that are not linear. Pipeline topologies more general than trees can require arbitrarily large pipe buffers, or worse, can cause deadlock. *Rc* has syntax for some kinds of non-linear but treelike pipelines. For example,

```
cmp <{old} <{new}
```

will regression test a new version of a command. < or > followed by a command in braces causes the command to be run with its standard output or input attached to a pipe. The parent command (`cmp` in the example) is started with the other end of the pipe attached to some file descriptor or other, and with an argument that will connect to the pipe when opened (e.g. `/dev/fd/6`.) On systems without `/dev/fd` or something similar (SunOS for example) this feature does not work.

## Exit status

When a command exits it returns status to the program that executed it. On Plan 9 status is a character string describing an error condition. On normal termination it is empty.

*Rc* captures commands' exit statuses in the variable `$status`. For a simple command the value of `$status` is just as described above. For a pipeline `$status` is set to the concatenation of the statuses of the pipeline components with | characters for separators.

*Rc* has a several kinds of control flow, many of them conditioned by the status returned from previously executed commands. Any `$status` containing only 0's and |'s has boolean value *true*. Any other status is *false*.

## Command grouping

A sequence of commands enclosed in { } may be used anywhere a command is required. For example:

```
{sleep 3600;echo 'Time''s up!'}&
```

will wait an hour in the background, then print a message. Without the braces:

```
sleep 3600;echo 'Time''s up!'&
```

this would lock up the terminal for an hour, then print the message in the background!

## Control flow – `for`

A command may be executed once for each member of a list by typing, for example:

```
for(i in printf scanf putchar)
   look $i /usr/td/lib/dw.dat
```

This looks for each of the words `printf`, `scanf` and `putchar` in the given file. The general form is

`for(`*name* `in` *list*`)` *command*

or

`for(`*name*`)` *command*

In the first case *command* is executed once for each member of *list* with that member assigned to variable *name*. If `in` *list* is not given, `$*` is used.

## Conditional execution – `if`

*Rc* also provides a general if-statement. For example:

```
if(cyntax *.c) cc -g -o cmd *.c
```

runs the C compiler whenever `cyntax` finds no problems with `*.c`. An "if not" statement provides a two-tailed conditional. For example:

```
for(i){
    if(test -f /tmp/$i) echo $i already in /tmp
    if not cp $i /tmp
}
```

This loops over each file in `$*`, copying to `/tmp` those that do not already appear there, and printing a message for those that do.

## Control flow – `while`

*Rc*'s while statement looks like this:

```
while(newer subr.c subr.o) sleep 5
```

This waits until `subr.o` is newer than `subr.c` (presumably because the C compiler finished with it).

## Control flow – `switch`

*Rc* provides a switch statement to do pattern-matching on arbitrary strings. Its general form is

```
switch(word){
case pattern ...
    commands
case pattern ...
    commands
...
}
```

*Rc* attempts to match the word against the patterns in each case statement in turn. Patterns are the same as for filename matching, except that / and the first characters of . and .. need not be matched explicitly.

If any pattern matches, the commands following that case up to the next case (or the end of the switch) are executed, and execution of the switch is complete. For example,

```
switch($#*){
case 1
    cat >>$1
case 2
    cat >>$2 <$1
case *
    echo 'Usage: append [from] to'
}
```

is an append command. Called with one file argument, it tacks standard input to its end. With two, the first is appended to the second. Any other number elicits a usage message.

The built-in '~' command also matches patterns, and is often more concise than a switch. Its arguments are a string and a list of patterns. It sets `$status` to true if and only if any of the patterns matches the string. The following example processes option arguments for the *man*(1) command:

```
opt=()
while(~ $1 -* [1-9] 10){
    switch($1){
    case [1-9] 10
        sec=$1 secn=$1
    case -f
        c=f s=f
    case -[qwnt]
        cmd=$1
    case -T*
        T=$1
    case -*
        opt=($opt $1)
    }
    shift
}
```

## Functions

Functions may be defined by typing

`fn` *name* { *commands* }

Subsequently, whenever a command named *name* is encountered, the remainder of the command's argument list will assigned to `$*` and *rc* will execute the *commands*. The value of `$*` will be restored on completion. For example:

```
fn g {
    gre -e $1 *.[hcyl]
}
```

defines `g` *pattern* to look for occurrences of *pattern* in all program source files in the current directory.

Function definitions are deleted by writing

`fn` *name*

with no function body.

## Command execution

Up to now we've said very little about what *rc* does to execute a simple command. If the command name is the name of a function defined

using `fn`, the function is executed. Otherwise, if it is the name of a built-in command, the built-in is executed directly by *rc*. Otherwise, if the name contains a `/`, it is taken to be the name of a binary program and is executed using *exec*(2). If the name contains no `/`, then directories mentioned in the variable `$path` are searched until an executable file is found.

## Built-in commands

Several commands are executed internally by *rc* because they are difficult or impossible to implement otherwise.

. [-i] *file* ...
: Execute commands from *file*. `$*` is set for the duration to the reminder of the argument list following *file*. `$path` is used to search for *file*. Option `-i` indicates interactive input – a prompt (found in `$prompt`) is printed before each command is read.

builtin *command* ...
: Execute *command* as usual except that any function named *command* is ignored. For example,

```
fn cd{
    builtin cd $* && pwd
}
```

defines a replacement for the `cd` built-in (see below) that announces the full name of the new directory.

cd [*dir*]
: Change the current directory to *dir*. The default argument is `$home`. `$cdpath` is a list of places in which to search for *dir*.

eval [*arg* ...]
: The arguments are catenated separated by spaces into a string, read as input to *rc*, and executed. For example,

```
x='$y'
y=Doody
eval echo Howdy, $x
```

would echo

```
Howdy, Doody
```

since the arguments of `eval` would be

```
echo Howdy, $y
```

after substituting for `$x`.

shift [*n*]
: Delete the first *n* (default 1) elements of `$*`.

wait [*pid*]
: Wait for the process with the given *pid* to exit. If no *pid* is given, all outstanding processes are waited for.

whatis *name* ...
: Print the value of each *name* in a form suitable for input to *rc*. The output is an assignment to a variable, the definition of a function, a call to `builtin` for a built-in command, or the path name of a binary program. For example,

```
whatis path g cd who
```

might print

```
path=(. /bin /usr/bin)
fn g {gre -e $1 *.[hycl]}
builtin cd
/bin/who
```

~ *subject pattern* ...
: The *subject* is matched against each *pattern* in turn. On a match, `$status` is set to true. Otherwise, it is set to `'no match'`. Patterns are the same as for filename matching. The *patterns* are not subjected to filename replacement before the '~' command is executed, so they need not be enclosed in quotation marks, unless of course, a literal match for `*` `[` or `?` is required. For example

```
~ $1 ?
```

matches any single character, whereas

```
~ $1 '?'
```

only matches a literal question mark.

## Advanced I/O Redirection

*Rc* allows redirection of file descriptors other than 0 and 1 (standard input and output) by specifying the file descriptor in square brackets `[ ]` after the < or >. For example,

```
cc junk.c >[2]junk.diag
```

saves the compiler's diagnostics in `junk.diag`.

File descriptors may be replaced by a copy, in the sense of *dup*(2), of an already-open file by typing, for example

```
cc junk.c >[2=1]
```

This replaces file descriptor 2 with a copy of file descriptor 1. It is more useful in conjunction with other redirections, like this

```
cc junk.c >junk.out >[2=1]
```

Redirections are evaluated from left to right, so this redirects file descriptor 1 to `junk.out`, then points file descriptor 2 at the same file. By contrast,

```
cc junk.c >[2=1] >junk.out
```

Redirects file descriptor 2 to a copy of file descriptor 1 (presumably the terminal), and then directs file descriptor 1 at a file. In the first case, standard and diagnostic output will be intermixed in `junk.out`. In the second, diagnostic output will appear on the terminal, and standard output will be sent to the file.

File descriptors may be closed by using the duplication notation with an empty right-hand side. For example,

```
cc junk.c >[2=]
```

will discard diagnostics from the compilation.

Arbitrary file descriptors may be sent through a pipe by typing, for example

```
cc junk.c |[2] grep -v '^$'
```

This deletes those ever-so-annoying blank lines from the C compiler's output. Note that the output of `grep` still appears on file descriptor 1.

Very occasionally you may wish to connect the input side of a pipe to some file descriptor other than zero. The notation

```
cmd1 |[5=19] cmd2
```

creates a pipeline with `cmd1`'s file descriptor 5 connected through a pipe to `cmd2`'s file descriptor 19.

## Here documents

*Rc* procedures may include data, called "here documents", to be provided as input to commands, as in this version of the *tel* command

```
for(i) grep $i <<!
...
nls 2T-402 2912
norman 2C-514 2842
pjw 2T-502 7214
...
!
```

A here document is introduced by the redirection symbol <<, followed by an arbitrary eof marker (`!` in the example). Lines following the command, up to a line containing only the eof marker are saved in a temporary file that it connected to the command's standard input when it is run.

*Rc* does variable substitution in here documents. The following *subst* command:

```
ed $3 <<EOF
g/$1/s//$2/g
w
EOF
```

changes all occurrences of `$1` to `$2` in file `$3`. To include a literal `$` in a here document, type `$$`. If the name of a variable is followed immediately by `^`, the caret is deleted.

Variable substitution can be entirely suppressed by enclosing the eof marker following << in quotation marks.

Here documents may be provided on file descriptors other than 0 by typing, for example

```
cmd <<[4]End
...
End
```

## Signals

*Rc* scripts normally terminate when an interrupt is received from the terminal. A function with the name of a signal, in lower case, is defined in the usual way, but called when *rc* receives the signal. Signals of interest are:

sighup
: Hangup. The controlling teletype has disconnected from *rc*.

sigint
: The interrupt character (usually ASCII del) was typed on the controlling terminal.

sigquit
: The quit character (usually ASCII fs, ctrl-\) was typed on the controlling terminal.

sigterm
: This signal is normally sent by *kill*(1).

sigexit
: An artificial signal sent when *rc* is about to exit.

As an example,

```
fn sigint{
    rm /tmp/junk
    exit
}
```

sets a trap for the keyboard interrupt that removes a temporary file before exiting.

Signals will be ignored if the signal routine is set to {}. Signals revert to their default behavior when their handlers' definitions are deleted.

## Environment

The environment is a list of name-value pairs made available to executing binaries. On Plan 9, the environment is stored in a file system named `#e`, normally mounted on `/env`. The value of each variable is stored in a separate file, with components terminated by ASCII nulls. (This is not quite as horrendous as it sounds, the file system is maintained entirely in core, so no disk or network access is involved.) The contents of `/env` are shared on a per-process group basis – when a new process group is created it effectively attaches `/env` to a new file system initialized with a copy of the old one. A consequence of this organization is that commands can change environment entries and see the changes reflected in *rc*.

There is not currently a way on Plan 9 to place functions in the environment, although this could easily done by mounting another instance of `#e` on another directory. The problem is that currently there can be only one instance of `#e` per process group.

## Local Variables

It is often useful to set a variable for the duration of a single command. An assignment followed by a command has this effect. For example

```
a=global
a=local echo $a
echo $a
```

will print

```
local
global
```

This works even for compound commands, like

```
f=/fairly/long/file/name {
    { wc $f; spell $f; diff $f.old $f } |
      pr -h 'Facts about '$f | lp -ddp
}
```

## Examples – *cd, pwd*

Program 1 shows a pair of functions that provide enhanced versions of the standard `cd` and `pwd` commands. (Thanks to Rob Pike for these.)

```
ps1='% '      # default prompt
tab='	'     # a tab character
fn pbd{
  /bin/pwd|sed 's;.*/;;'
}
fn cd{
  builtin cd $1 &&
  switch($#*){
  case 0
    dir=$home
    prompt=($ps1 $tab)
  case *
    switch($1)
    case /*
      dir=$1
      prompt=('{pbd}^$ps1 $tab)
    case */* ..*
      dir=()
      prompt=('{pbd}^$ps1 $tab)
    case *
      dir=()
      prompt=($1^$ps1 $tab)
    }
  }
}
fn pwd{
  if(~ $#dir 0)
    dir='{/bin/pwd}
  echo $dir
}
```

**Programe 1:** *cd, pwd*

Function `pwd` is a version of the standard `pwd` that caches its value in variable `$dir`, because the genuine `pwd` can be quite slow to execute.

Function `pbd` is a helper that prints the last component of a directory name. Function `cd` calls the `cd` built-in, and checks that it was successful. If so, it sets `$dir` and `$prompt`. The prompt will include the last component of the current directory (except in the home directory, where it will be null), and `$dir` will be reset either to the correct value or to `()`, so that the `pwd` function will work correctly.

## Examples – *man*

The *man* command prints pages from of the Programmer's Manual. It is called, for example, as

```
man 3 isatty
man rc
man -t cat
```

In the first case, the page for *isatty* in section 3 is printed. In the second case, the manual page for *rc* is printed. Since no manual section is specified, all sections are searched for the page, and it is found in section 1. In the third case, the page for *cat* is typeset (the `-t` option). Program 2 shows the *man* command.

```
cd /n/bowell/usr/man || {
  echo $0: Manual not on line! >[1=2]
  exit 1
}
NT=n  # default nroff
s='*' # section, default try all
for(i) switch($i){
case -t
  NT=t
case -n
  NT=n
case -*
  echo Usage: $0 '[-nt] [section] page ...' >[1=2]
  exit 1
case [1-9] 10
  s=$i
case *
  eval 'pages=man'$s/$i'.*'
  for(page in $pages){
    if(test -f $page)
      $NT^roff -man $page
    if not
      echo $0: $i not found >[1=2]
  }
}
```

**Program 2:** *The* `man` *command*

Note the use of `eval` to make a list of candidate manual pages. Without `eval`, the `*` stored in `$s` would not trigger filename matching – it's enclosed in quotation marks, and even if it weren't, it would be expanded when assigned to `$s`. Eval causes its arguments to be re-processed by *rc*'s parser and interpreter, effectively delaying evaluation of the `*` until the assignment to `$pages`.

## Examples – *holmdel*

Program 3 is an *rc* script that plays the deceptively simple game *holmdel*, in which the players alternately name Bell Labs locations, the winner being the first to mention Holmdel.

```
t=/tmp/holmdel$pid
fn read{
	$1='{awk '{print;exit}'}
}
ifs='
'	# just a newline
fn sigexit sigint sigquit sighup{
	rm -f $t
	exit
}
cat <<'!' >$t
Allentown
Atlanta
Cedar Crest
Chester
Columbus
Elmhurst
Fullerton
Holmdel
Indian Hill
Merrimack Valley
Morristown
Piscataway
Reading
Short Hills
South Plainfield
Summit
Whippany
West Long Branch
!
while(true){
   lab='{/usr/games/fortune $t}
   echo $lab
   if(~ $lab Holmdel){
      echo You lose.
      exit
   }
   while(read lab;
      ! grep -i -s $lab $t)
      echo No such location.
   if(~ $lab [hH]olmdel){
      echo You win.
      exit
   }
}
```

**Program 3:** `holmdel`

This script is worth describing in detail (rather, it would be if it weren't so silly.)

Variable `$t` is an abbreviation for the name of a temporary file. Including `$pid`, initialized by *rc* to its process-id, in the names of temporary files insures that their names won't collide, in case more than one instance of the script is running at a time.

Function `read`'s argument is the name of a variable into which a line gathered from standard input is read. `$ifs` is set to just a newline. Thus `read`'s input is not split apart at spaces, but the terminating newline is deleted.

A handler is set to catch `sigint`, `sigquit`, and `sighup`, and the artificial `sigexit` signal. It just removes the temporary file and exits.

The temporary file is initialized from a here document containing a list of Bell Labs locations, and the main loop starts.

First, the program guesses a location (in `$lab`) using the `fortune` program to pick a random line from the location list. It prints the location, and if it guessed Holmdel, prints a message and exits.

Then it uses the `read` function to get lines from standard input and validity-check them until it gets a legal name. Note that the condition part of a `while` can be a compound command. Only the exit status of the last command in the sequence is checked.

Again, if the result is Holmdel, it prints a message and exits. Otherwise it goes back to the top of the loop.

## Discussion

Steve Bourne's `/bin/sh` is extremely well-designed; any successor is bound to suffer in comparison. I have tried to fix its best-acknowledged shortcomings and to simplify things wherever possible, usually by omitting unessential features. Only when irresistibly tempted have I introduced novel ideas. Obviously I have tinkered extensively with Bourne's syntax, that being where his work was most open to criticism.

The most important principle in *rc*'s design is that it's not a macro processor. Input is never scanned more than once by the lexical and syntactic analysis code (except, of course, by the `eval`

command, whose *raison d'etre* is to break the rule).

Bourne shell scripts can often be made to run wild by passing them arguments containing spaces. These will be split into multiple arguments using `IFS`, often at inopportune times. In *rc*, values of variables, including command line arguments, are not re-read when substituted into a command. Arguments have presumably been scanned in the parent process, and ought not to be re-read.

Why does Bourne re-scan commands after variable substitution? He needs to be able to store lists of arguments in variables whose values are character strings. If we eliminate re-scanning, we must change the type of variables, so that they can explicitly carry lists of strings.

This introduces some conceptual complications. We need a notation for lists of words. There are two different kinds of concatenation, for strings – `$a^$b`, and lists – `($a $b)`. The difference between `()` and `''` is confusing to novices, although the distinction is arguably sensible – a null argument is not the same as no argument.

Bourne also rescans input when doing command substitution. This is because the text enclosed in back-quotes is not properly a string, but a command. Properly, it ought to be parsed when the enclosing command is, but this makes it difficult to handle nested command substitutions, like this:

```
size=`wc -l \`ls -t|sed 1q\``
```

The inner back-quotes must be escaped to avoid terminating the outer command. This can get much worse than the above example; the number of \'s required is exponential in the nesting depth. *Rc* fixes this by making the backquote a unary operator whose argument is a command, like this:

```
size=`{wc -l `{ls -t|sed 1q}}
```

No escapes are ever required, and the whole thing is parsed in one pass.

For similar reasons *rc* defines signal handlers as though they were functions, instead of associating a string with each signal, as Bourne does, with the attendant possibility of getting a syntax error message in response to typing the interrupt character. Since *rc* parses input when typed, it reports errors when you make them.

For all this trouble, we gain substantial semantic simplifications. There is no need for the distinction between `$*` and `$@`. There is no need for four types of quotation, nor the extremely complicated rules that govern them. In *rc* you use quotation marks exactly when you want a syntax character to appear in an argument. `IFS` is no longer used, except in the one case where it was indispensable: converting command output into argument lists during command substitution.

This also avoids an important security hole [Ree88a]. *System*(3) and *popen*(3) call `/bin/sh` to execute a command. It is impossible to use either of these routines with any assurance that the specified command will be executed, even if the caller of *system* or *popen* specifies a full path name for the command. This can be devastating if it occurs in a set-userid program. The problem is that `IFS` is used to split the command into words, so an attacker can just set `IFS=/` in his environment and leave a Trojan horse named `usr` or `bin` in the current working directory before running the privileged program. *Rc* fixes this by not ever rescanning input for any reason.

Most of the other differences between *rc* and the Bourne shell are not so serious. I eliminated Bourne's peculiar forms of variable substitution, like

```
echo ${a=b} ${c-d} ${e?error}
```

because they are little used, redundant and easily expressed in less abstruse terms. I deleted the builtins `export`, `readonly`, `break`, `continue`, `read`, `return`, `set`, `times` and `unset` because they seem redundant or only marginally useful.

Where Bourne's syntax draws from Algol 68, *rc*'s is based on C or Awk. This is harder to defend. I believe that, for example

```
if(test -f junk) rm junk
```

is better syntax than

```
if test -f junk; then rm junk; fi
```

because it is less cluttered with keywords, it avoids the semicolons that Bourne requires in odd places, and the syntax characters better set off the active parts of the command.

The one bit of large-scale syntax that Bourne unquestionably does better than *rc* is the `if` statement with `else` clause. *Rc*'s `if` has no terminating `fi`-like bracket. As a result, the parser cannot tell whether or not to expect an `else` clause without looking ahead in its input. The problem is that after reading, for example

```
if(test -f junk) echo junk found
```

in interactive mode, *rc* cannot decide whether to execute it immediately and print `$prompt(1)`, or to print `$prompt(2)` and wait for the `else` to be typed. In the Bourne shell, this is not a problem, because the `if` command must end with `fi`, regardless of whether it contains an `else` or not.

*Rc*'s admittedly feeble solution is to declare that the `else` clause is a separate statement, with the semantic proviso that it must immediately follow an `if`, and to call it `if not` rather than `else`, as a reminder that something odd is going on. The only noticeable consequence of this is that the braces are required in the construction

```
for(i){
    if(test -f $i) echo $i found
    if not echo $i not found
}
```

and that *rc* resolves the "dangling else" ambiguity in opposition to most people's expectations.

It is remarkable that in the four most recent editions of the UNIX system programmer's manual the Bourne shell grammar described in the manual page does not admit the command `who|wc`. This is surely an oversight, but it suggests something darker: nobody really knows what the Bourne shell's grammar is. Even examination of the source code is little help. The parser is implemented by recursive descent, but the routines corresponding to the syntactic categories all have a flag argument that subtly changes their operation depending on the context. *Rc*'s parser is implemented using *yacc*, so I can say precisely what the grammar is.

Its lexical structure is harder to describe. I would simplify it considerably except for two things. There is a lexical kludge to distinguish between parentheses that immediately follow a word with no intervening spaces and those that don't that I would eliminate if there were a reasonable pair of characters to use for subscript brackets. I could also eliminate the insertion of free carets if users were not adamant about it.

## Acknowledgements

Rob Pike, Howard Trickey and other Plan 9 users have been insistent, incessant sources of good ideas and criticism. Some examples in this document are plagiarized from [Bou78a], as are most of *rc*'s good features.

## References

[Bou78a] S. R. Bourne, "UNIX Time-Sharing System: The UNIX Shell", *Bell System Technical Journal* **57**(6), pp. 1971-1990 (July-August 1978).

[Ree88a] J. Reeds, "`/bin/sh`: the biggest UNIX security loophole", 11217-840302-04TM, AT&T Bell Laboratories (1988).

# The SSBA at AFUU: A Progress Report

*C.Binot, P.Dax, N.Doduc, M.Gaudet*
*binot@afuu.fr, dax@enst.fr, ndoduc@framentec.fr*

*AFUU*

The SSBA by the AFUU Benchmark group is 3 years old and yet very active and flourishing. Here is a progress report focusing on the 1989-1990 time frame as well as on the planned evolution toward a next release.

## The SSBA: the background

The popularisation of the UNIX(tm) workstation and the recognition of UNIX(tm) as the most important operating system in the supercomputer arena as well as the feeling that it is potentially important in the conservative business-processing market, all proved to be a most appropriate ground for Benchmarking activities.

Created circa March 87, the Association Francaise des Utilisateurs d'UNIX(tm) et des Systemes Ouverts (AFUU) Benchmark group of reflects users' needs in this domain. The group wanted to provide users and consumers with a valuable and practical tool in evaluating basic characteristics of, initially, UNIX-running on personal workstations. This very clearly set goal was met quite enthusiastically by many people from different horizons, mainly users from big accounts or academia, but strangely enough, also support-staff people from French subsidiaries of American computer manufacturers for whom the group is an opportunity to understand what their mother companies are cooking and have them eat.

The work progressed the usual way thanks to the very dynamic atmosphere surrounding AFUU, and then by June '88, the very first version, 1.0, of the SSBA was fielded for tests.

With respect to EUUG, and in due time, the Benchmark group announced its creation and its activity in this Newsletter (Benchmarking in the AFUU, Vol. 7, no. 4, Winter 1987) and then made its first status report at the 1989 Spring (Bruxelles) EUUG Conference. Here, we are putting a kind of progress report concerning not only the SSBA but also the Benchmark group, and regarding the last 12 months.

## La SSBA en France!

Within France, the SSBA has established itself as a mandatory item in many request-for-tender (*); many important accounts routinely request its inclusion amongst other tests or its certification (by AFUU) results, leading to a very widespread use. One vendor from a big manufacturer has even confided to us that, in 1989 alone, he has got more than 30 requests (for SSBA results) from his clients and prospects! Our statistical data for the Jan-Oct 1989 period showed that, out of 383 PD-software tapes purchased from AFUU, 89, ie 23%, are SSBA tapes. We even believe that most computer manufacturers do have a copy of the SSBA, usually and openly supplied by their French offices.

Thus the SSBA may get the credit for a strong surge of interest in the French professional media in general and magazines particularly, for this very delicate and sensitive subject. There are many points of merit to be told.

As early as the summer of 1988, Le Centre d'Experimentation du Logiciel made a survey about then existing workstations: the SSBA is used as **the** tool for this survey. Since then, at least two other similar studies have occurred and of course the SSBA has also proven its appropriateness there as well.

The display in Tribunix, AFUU's Newsletter, of the approved results began with the issue #27, Vol. 5, Jui-Aou/89 and from then on, became a regular column in Tribunix. At the beginning of 1990, we released the first special Benchmark issue of Tribunix, issue #30, Jan-Feb/90, with many contributions from people interested in this subject. Then at the conjunction of the UNIX Convention, March 1990, and the third anniversary of the Benchmark group, we built out a Dossier Benchmark Tribunix (**), that

(*) thanks to the wise and efficient lobbying of some of us ...

(**) available from AFUU secretariat.

contained selected articles from the previous issues, and most notably, all the SSBA results (about 60) collected to date. All these printed items get such a favourable welcome from many quarters that, right now, we're considering keeping them as regular items within Tribunix.

Furthermore, along with internal informational activities and beside the regular meetings (once every 6 or 7 weeks), a regular and permanent column is maintained in our Tribunix, whose name is **La Chronique des Benchmarks**, starting with issue #28, Sep-Oct/89. We keep ourselves fully informed of the happenings of different benchmarking activities (eg. when known of course, we're no newsmen and do not have their resources, although we're helped frequently by those newsmen) such as: SPEC, the Perfect Club, Utrecht's SSB ... as well as, say, InfoPC evaluation of numeric coprocessors (for i386) ...

The Benchmark group also has a rôle of a librarian regarding Benchmark software: we collect and distribute public-domain Benchmark source. Very recently, this role has been extended to articles and papers although this latter part is even more amateurish, at least at the moment, then the previous one.

## The SSBA elsewhere ...

On the external front, the group and the SSBA are understandably very active too: many of its members has close contacts with corresponding entities either within or outside Europe.

The Benchmark group as is does have contacts with similar organisations like SPEC, the Performance Measurement group of UI ... and of course many of us have links witb peoples working in internal Performance groups of many manufacturers.

We succeeded in having a member of the SPEC Steering Committee come and give a talk at our UNIX Convention (March '90).

The SSBA is used in product evaluation articles by IX Multiuser-Multitasking-Magazine (Hannover, Germany) and the Mai-Jun/90 issue dealt with, among many machines, the IBM RS/6000.

The SSBA is public-domain software: just ask for the EUUGD20 tape from EUUG Software Distribution. It's use is of course completely free but the publication of the results in Tribunix needs agreement from the Benchmark group.

Lastly, many copies have been sent to many educational institutions throughout Europe (Dortmund, Karlsruhe, Louvain, Liege, Valencia, ... or even to America: Worcester ...) to be used either as one of many benchmark sets to be examined within many Computer Performance courses or simply as a tool for purchase evaluation. Lately, the SSBA has also made contacts with Eastern European countries (***) and also has taken seed across the Mediterranean: Algeria, the later case within the framework of AFUU helping creating a local UNIX user group. The next release(s)

The SSBA version 1.21 has remained unchanged since Feb. 89 and in fact is **the** SSBA everybody mentioned. Having been proven in many enduring circumstances, resulting in many comments and much feed-back (****) the SSBA is now ripe for timely evolution. A Road-Map is being circulated among members for consultation and toward convergence for changes. It is most likely that before Christmas, a revised version will be released, while the other new suite of the SSBA family, although well on track, will not be ready before Spring 1991.

The actual version, 1.21, deals mainly with basic, eg. systems-level, performance will be revised and may get or include the following items: Dhrystone version 2 replacing version 1, an officially maintained version Whetstone (?), new disk throughput component(s), modifications of the SSBA's version of the Musbus ... and probably a small amount of multiprocessing measurement as well ...

The SSBA has ambitions to be a family of many suites: the version 1.x as told previously deals with generic and raw systems performance. The next suite(s) will be application-oriented, and while many domains being of immense and immediate interests, such as Real time, Graphics, Network ... are considered, we are very conservative in looking at the content of our next suite. However, it is not imprudent to state that we are quite interested, and more or less advanced in our evaluation, in items such as IOBench

(***) thanks to the *perestroika* ! Long live the *glasnost* !

(****) as well as bug reports, heated discussions and email traffic congestion.

(Wolman & Olson, Prime), Xbench (Gittinger, Siemens) ...

Another very interesting point is the analysis and use of these *raw* data obtained. Up to now, we have strived not to succumb to the temptation of giving a *single value of merit* to each machine. Recently, we have enrolled the Laboratoire de Statistiques et Probabilities de l'Universite de Toulouse to help us in processing our data. Initial results are very encouraging and we think that a new dimension of utility can and will be added to our valuable results.

The very last item in early discussion is the moving toward an EUUG-working group as suggested to and by the EC: may we wish this a real success so that the next progress report will be the first of a series of the (Euro-)Benchmark (Super-)group? **Benchmarkeurs de tous pays, unissons-nous !**

## About the authors:

*Christophe Binot* started his commitment in UNIX when he worked at Universite de Valenciennes on MMI, and having to *choose* his first workstations, decided to invest in the hazardous ground of Workstation, Benchmarking, UNIX Culture ... all these are the names of the AFUU working groups that he chairs. When times and his AFUU vice-chairman's job permit, he also is a full-paid support engineer for HP-France. In the old times, when still at Valenciennes, Christophe many times visited his Sun 3/260 at 2 AM without proper authorisation from his young wife. (tel: +33-145916854, email: binot@afuu.fr).

*Philippe Dax* is an antique figure of a very old nevertheless prestigious engineering school (Sup. Telecom., the Paris establishment! please) where he used to take care of, among many things, *schoene fraulein* and *gentilles jeunes etudiantes* querying about make, yacc ... and a not-so-small set of Sun workstations. Another not-so-small activity of him is the AFUU's newsletter Tribunix that keeps growing in its scope, its thickness and the times it takes from its editor-in-chief. Philippe has put altogether, consciously, most of the SSBA script and, unintentionally, some of the yet-to-be-discovered bugs. (tel: +33-145817687; email: dax@enst.fr).

*Nhuan Doduc* makes his living by working in DP since his leaving theoretical physics, and makes his pleasure by benchmarking a not-too-unknown FORTRAN program since very long time ago. Nhuan is a strong driving force within many AFUU's groups (Calculateurs Scientifiques ...), spends much of his time in contributing to Philippe's Tribunix, participates actively in the G.U.S, (aka. the French S.U.G.) and the Workstation group of the CUBE, (the French Bull user group) ... and, for what is left of his time, also officially works for Framentec and Cognitech, two expert-system software houses. Nhuan is the *keeper* of the benchmark library of the group. (tel: +33-147964633; email: ndoduc@framentec.fr).

*Michel Gaudet* stills work in Laboratoire de Biorheologie Hydrodynamique PhysicoChimique of Universite Paris VII, where he reigns supreme on many PCs openly connected (eg. cables disconnected most of the times) to a 4Mb Sun 3/140. Beyond UNIX, Michel's many expertises include the PC hardware, Ms/Dos, the nonfunctioning of the SSBA's Musbus, the I/O rates from the Saxer as well as from many other I/O benchs, but then he is still hesitant on deciding to move his Sun to OS 4.1, that is with as many as 4 (four) Mb of Ram. While waiting for 4.1 to arrive, he makes lots of travels to take care of most of the SSBA results. (tel: +33-143362525 ext 4333).

## Request for Test

The AFUU has decided to make an open way in which the UNIX users' community can access the computer performance information printed in Tribunix.

With this in mind the AFUU has offere those producing and distributing UNIX based systems to publish in Tribunix, under their own name and with the AFUU certificate of testing, the output obtained from running the SSBA 1.21F, with the aim of informing the membership of the AFUU. A request to print this in Tribunix must be supported by a certificate of having run the benchmark in the presence of a person approved by AFUU for this purpose (Nhuan Doduc, Philippe Dax, Michel Gaudet). Also needed is the location of the test, electronic evidence, and the output of the test.

*Dominique Maisonneuve, Christophe Binot, Nhuan Doduc, Philippe Dax.*

An example of test output follows:

| SYNTHESE DES RESULTATS DE LA SSBA 1.21F (27/02/89) | |
|---|---|
| **Identification : BULL DPX/2 340 Moto 68030/68882 B.O.S. 1.1 33 Mhz 16 Mbytes 2*300 Mbytes** | |
| **ENVIRONEMENT** | |
| Type of Unix | SVR3 |
| Command C | cc -FO -DTERMIO -DSysV |
| Command F | f77 -FO -DTERMIO -DSysV |
| Name of site | B.O.S. |
| Name of login of benchmarker | root |
| Value of HZ | 100 |
| Number of processors before start | 6 |
| Number of processors staying for the execution | 251 |
| Available virtual memory space | 0 Mb |
| **RESULTS** | |
| Date of start | Thu May 10 11:56:03 EET 1990 |
| Mips/Joy | 5.63636 |
| Dhrystone (without registers,without optimisation) | 11483 Dhry/s |
| Dhrystone (with registers,without optimisation) | 11483 Dhry/s |
| Dhrystone (without registers,with optimisation) | 12860 Dhry/s |
| Dhrystone (with registers,with optimisation) | 12879 Dhry/s |
| Whetstone (single precision) | 2867 KWhet/s |
| Whetstone (double precision) | 2637 KWhet/s |
| Linpack (single precision, rolled) | 330 Kflops |
| Linpack (single precision, unrolled) | 341 Kflops |
| Linpack (double precision, rolled) | 274 Kflops |
| Linpack (double precision, unrolled) | 280 Kflops |
| Utah (execution time) | user=12.22s, sys=2.8s, u+s=15.02s |
| Outils (execution time) | user=52.84s, sys=7.9s, u+s=60.74s |
| Byte (execution time) | user=2.28s, sys=11.96s, u+s=14.24s |
| Saxer (thoughput W/S disc) | 548.24 Kb/s |
| Testc (execution time) | user=66.30s, sys=1.72s, u+s=68.02s |
| Doduc (double precision) | iterations=5491, temps=946.18s, ratio=51 |
| Bsd memory (execution time) | user=82.74s, sys=1.72s, u+s=84.46s |
| Bsd syscalls (execution time) | user=0.64s, sys=16.98s, u+s=17.62s |
| Bsd pipes (execution time) | user=0.84s, sys=43.12s, u+s=43.96s |
| Bsd forks/execs (execution time) | user=3.2s, sys=214.74s, u+s=217.94s |
| Musbus (thoughput W/S disc for 62 blocks) | write=1033.3 Kb/s read=1550.0 Kb/s copy=620.0 Kb/s |
| Musbus (thoughput W/S disc for 125 blocks) | write=446.4 Kb/s read=2083.3 Kb/s copy=420.2 Kb/s |
| Musbus (thoughput W/S disc for 250 blocks) | write=740.7 Kb/s read=1785.7 Kb/s copy=554.5 Kb/s |
| Musbus (thoughput W/S disc for 500 blocks) | write=979.6 Kb/s read=1785.7 Kb/s copy=306.1 Kb/s |
| Musbus (thoughput W/S disc for 1000 blocks) | write=1173.7 Kb/s read=503.4 Kb/s copy=139.8 Kb/s |
| Musbus real time | 794.00s |
| Musbus cpu time + system | 665.37s |
| Average size of an executable | 39.8967 Kb |
| Total Time | 5977s |
| Number of cycles | 78 |
| Average number of users | 1 |
| Average number of processors | 17 |
| Date of end | Thu May 10 13:35:40 EET 1990 |

**COMMENTAIRES**

Passage chez : BULL MTS - 1 rue of Provence - Parc Surieux BP208 - 38432 Echirolles Ceofx

*Marie-Hlne Marc et Michel Gauoft*

# USL Column

*Chris Schoettle*

*UNIX System Laboratories Europe*

For further information on this column please contact Gill Mogg on *gill@uel.uucp*. Gill is Market Communications Manager at USLE. The guest writer this issue is Chris Schoettle.

Chris Schoettle is a Senior Consultant with UNIX System Laboratories based in London, England. Mr. Schoettle has BA and MSc degrees in Computer Science and has worked extensively in the international UNIX systems market. He has presented the technical contents of UNIX System V Release 4.0 and of the OPEN LOOK Graphical User Interface thoughout Europe, including the EUUG, UNIForum, and USL's Software Developer Conferences. He represents USL on the X/Open User Interface Working Group.

## OPEN LOOK - A Consistent Approach to a GUI Architecture

Much has been written, discussed and even hyped on Graphical User Interfaces (GUIs) to the UNIX Operating System. It is becoming increasingly more difficult to grasp the underlying principles on which the products are based. Here, we will address the rationale that UNIX System Laboratories (USL) has followed in developing and licensing the software that implements the OPEN LOOK® GUI.

A GUI comprises three functional areas:

- a platform
- a programming interface
- a user perspective

Before attempting to define the above terms, it is worth drawing an analogy (albeit over-simplified). Consider a GUI as an automobile. The platform is the entire interworkings provided in the base vehicle. There might also be an optional extra such as an on-board computer. The programming interface of the computer can be thought of as that part which interoperates with the interworkings of the automobile.

The user perspective of the computer is what the driver sees on the face of the computer and how the driver operates it. In general terms, the programming interface of the automobile is how the base car and all the optional extras interoperate. The user perspective is everything that the driver sees and operates in the car, including the base and optional extras.

Now let's replace the world of automobiles with that of GUIs. The platform is the software which manages the window system (the base car), including the display and input device. Other software, such as a toolkit, can be provided on top of the platform (an optional extra), and is usually dependent on the platform on which it is implemented.

The programming interface is the syntax and semantics defined for use by applications interfacing with a platform and possibly a toolkit. A toolkit can provide functionality in addition to that provided by the platform; this can include components such as buttons, scrollbars and menus. For example, a toolkit may provide software to implement a scrollbar in an application. The programming interface for this scrollbar would most likely be a combination of the semantics defined by the platform and the syntax defined for the scrollbar in the toolkit.

The user perspective is what the end user will see and operate with on the terminal, irrespective of the underlying software; this is widely termed as the Look and Feel. In our example of the scrollbar, the user perspective of the scrollbar is what it actually looks like when seen on the terminal and how the user uses the mouse to

operate the scrollbar within an application.

In early GUIs, such as SunView* or the Macintosh*, the GUI is inherently part of the platform, operating system, and even hardware; these are kernel-based window systems. As the industry has moved towards the Open Systems marketplace, client-server based window systems have evolved as platforms. These provide an environment for networking and portability, across different operating systems and hardware, and as a base for different GUI architectures. Two client-server based window systems of note are the X Window System* and NeWS*, the network extensible Window System.

The X Window System, or X, is a defacto (and now X/Open official) standard, developed by the Massachusetts Institute of Technology. NeWS is a PostScript* based architecture developed by Sun Microsystems. The programming interface, unlike the user perspective, of a GUI for a platform can be dependent on that platform, in this case X or NeWS. Specifically with the OPEN LOOK GUI, while there is one user perspective, there are three different programming interfaces, different by both design and objective.

The OPEN LOOK X Toolkit (code named Xt+) is a widget set built at the Xt Intrinsics layer of X (this is just X terminology for saying that it is a set of components built on the top layer of the X platform). The programming interface for the OPEN LOOK X Toolkit is a combination of the syntax provided in the widgets of the OPEN LOOK X Toolkit and the semantics of X. It is the one which is most commonly compared to the OSF/Motif* Toolkit.

The OPEN LOOK XView* Toolkit has a programming interface very close to that of SunView, a kernel-based window system. It has been designed with the purpose of migrating existing SunView applications to X (and the Open Systems marketplace). To maintain the SunView programming interface, the OPEN LOOK XView Toolkit has been developed at the Xlib layer of X, a lower layer than the Xt Intrinsics.

The OPEN LOOK tNt (the NeWS Toolkit) has been developed in PostScript to provide a programming interface for software developers wanting to develop OPEN LOOK applications on NeWS.

Thus, for a software developer there are three different programming interfaces for the OPEN LOOK GUI. For the end user though, there is just one user perspective, OPEN LOOK. The Look and Feel for OPEN LOOK is strictly defined in a set of published books known as the OPEN LOOK Specification and Style Guides. This Look and Feel has gone through an extensive industry review process. It is published so that software developers have the opportunity of developing new toolkits which implement the Look and Feel of OPEN LOOK. Thus, there is the possibility of having additional programming interfaces (in addition to the three already mentioned) that have been designed to meet a market requirement for implementing the Look and Feel of OPEN LOOK. One such additional interface is the OI Library from Solbourne which has been designed for the C++ market and is discussed further below.

What a user reads in the OPEN LOOK Specification is exactly what the user will see on the terminal. The user will not be able to distinguish (and will probably not care) which platform and OPEN LOOK toolkit programming interface was used to develop the application. For the user it is all the same, OPEN LOOK.

Let's return now to OPEN LOOK product issues. OPEN LOOK GUI Release 2.0 is the most recent release of the OPEN LOOK X Toolkit. It builds on the capabilities provided in OPEN LOOK GUI Release 1.0, first released in March 1989. Without changing the programming interface, it provides the final Look and Feel defined in the OPEN LOOK Specification. It has reduced memory requirements and performance improvements with gadgets and flat widgets (more object-oriented by not forcing each component to be window). It is based on X Window System Version 11 Release 3 (X11R3); a later release will be based on X11R4. It has an end user environment which includes the applications Window Manager, Workspace Manager, Terminal Emulator, Pixmap Editor, Print Screen, and File Manager. The File Manager provides a graphical user interface to the UNIX System V file system and is the beginning of a desktop metaphor.

The X Window System has been productised by USL into XWIN. XWIN Release 3.0 is based on X11R3 (a later release will be based on X11R4) and contains many improvements over the public domain X from MIT. XWIN Release 3.0 has reduced memory requirements, many bug fixes, is implemented on STREAMS, uses shared libraries, provides a colour Terminal Emulator, and has

much improved performance.

NeWS has been released by USL in a merged window system architecture with X, called X11/NeWS*. Recognising that both X and NeWS are client-server based window systems with commonalities at the server level, X11/NeWS contains a single merged server for both X and NeWS. It can be thought of as a single room containing such items as a common window tree and event queue, but with two doors for entering, one for the X Protocol and one for the NeWS PostScript protocol.

The graphics products we have discussed are all available now. All are included in either the source code of UNIX System V Release 4, as a separate bundled graphics product called Graphics Services Version 2, or as separate individual products.

The OPEN LOOK software has a consistent approach to a GUI architecture. This is pictorially represented in Figure 1. Here we have a platform of UNIX System V Release 4 with XWIN and X11/NeWS. At the level of the programming interface, there are three OPEN LOOK toolkits. In developing an application, a software developer chooses the OPEN LOOK toolkit with the most suitable programming interface.

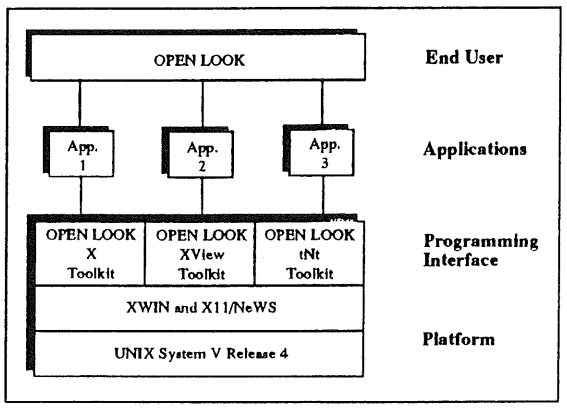



Figure 1: OPEN LOOK GUI Architecture

Application 1 is most likely a new application built at the Xt Intrinsics layer of X and so uses the OPEN LOOK X Toolkit. Application 2 has probably been recently converted from SunView and so uses the OPEN LOOK XView Toolkit. Application 3 could be a PostScript application on NeWS and so uses the OPEN LOOK tNt Toolkit. A different programming interface has been designed to meet the design objectives for each of the applications. From the end user perspective it is all the same, OPEN LOOK, just as it has been defined in the OPEN LOOK Specification. Two platforms, three toolkits, one Look and Feel - this is the consistent approach of OPEN LOOK.

Before leaving, it is worth looking at the future of OPEN LOOK. One of the most commonly asked questions is in comparison to the Motif Toolkit as to why USL doesnUt cancel OPEN LOOK and use Motif. This is a question we asked both ourselves and our customers in Autumn 1989. The internal USL answer is clear; we feel OPEN LOOK is a technically superior product with much to offer the Open Systems marketplace. We could go through a list of reasons here, but it would soon become clear that there are not many major differences to highlight. There are many smaller differences, but even these are becoming less obvious in ensuing releases of OPEN LOOK and Motif.

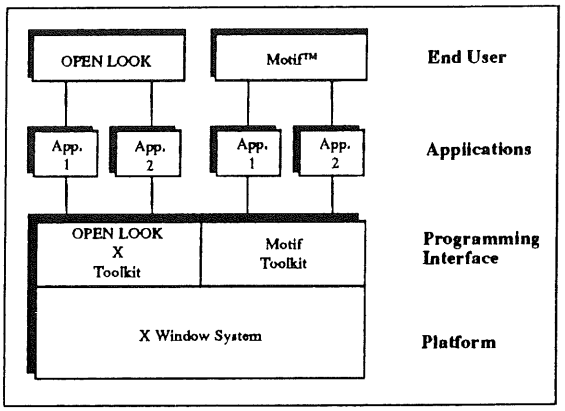



Figure 2

There are two primary differences that are worth looking at between OPEN LOOK and Motif. One is the different Look and Feel, but this is becoming less of an issue as the market is recognising that there are different market areas for different Look and Feels and that this is even desirable. The other difference is the programming interface. The three OPEN LOOK toolkits have three different programming interfaces by both design and objective. The OPEN LOOK X Toolkit and the Motif Toolkit are both widget sets at the Xt Intrinsics layer of X, and there is no need for them to have different programming interfaces. The OPEN LOOK X Toolkit and the Motif Toolkit do not have different programming interfaces by design and objective, they have different programming interfaces by accident.

The result is shown in Figure 2, where Applications 1 and 2 have been developed with the OPEN LOOK X Toolkit to use the OPEN LOOK Look and Feel. If there is a need for Applications 1 and 2 to use the Motif Look and Feel, they have to be ported to use the Motif programming interface. This is unacceptable to software developers and was agreed as the most significant problem facing the X/Open Desktop Computing Workgroup.

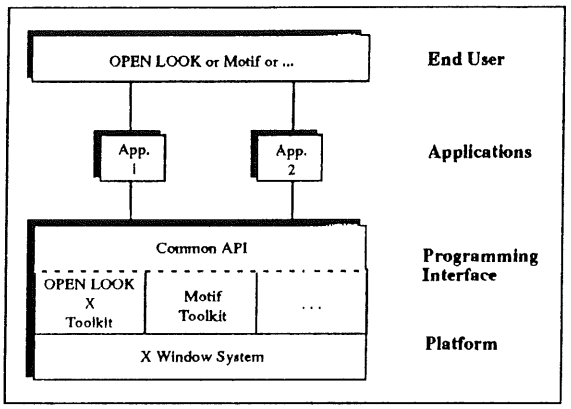



Figure 3

To help resolve this situation, the IEEE P1201 standards group is working on defining a Common API (Application Programming Interface) to OPEN LOOK and Motif. Pictorially, the result should be similar to Figure 3, where Application 1 and Application 2 have been written to a single Common API and can effectively toggle between the OPEN LOOK and Motif Look and Feels, without having to be ported. USL and UNIX International fully support a Common API and are helping to define a standard through the IEEE P1201 forum. We would welcome support from all interested parties.

While a Common API standard is being defined, several companies have already made the strategic decision of developing their own single programming interfaces incorporating both OPEN LOOK and Motif Look and Feels. One such company is Solbourne which has developed the OI Library, a single C++ interface which provides the Look and Feel of both OPEN LOOK and Motif. We consider software such as the OI Library from Solbourne as proof of existence that a Common API is definitely feasible.

Going back to the original question: why is USL continuing with the OPEN LOOK strategy? We have already said that from an internal USL perspective we feel it is a better product, but what do our customers think? After all, if our customers do not want a product, then it takes little market awareness to realise that we should not sell it. In Autumn 1989 we asked our customers what they thought about OPEN LOOK.

| | |
|---|---|
| • Shipping Product: | |
| AT&T CSB | Quest Systems Corp. (S/ware) |
| AT&T USL (Software) | Solbourne Computer, Inc. |
| Harris Corporation | Sun Microsystems, Inc. |
| Quantum Software Systems Ltd | |
| • Commitment: | |
| Altos India | Labtam |
| Amdahl | Motorola |
| ARIX | Olivetti |
| Commodore | Pyramid |
| Fujitsu | Sequoia |
| HCL Ltd. | SONY |
| ICL | Tolerant Software (Software) |
| Intel | TOSHIBA |
| INTERACTIVE (Software) | Wipro |
| KonsultHusetData (Software) | |

Figure 4: OPEN LOOK GUI OEM Commitment

The answer has been a resounding response in favour of OPEN LOOK. USL is keeping OPEN LOOK because our customers asked us to. In Figure 4 is a list of OEMs who have committed to ship OPEN LOOK. In Figure 5 is a list of ISVs who have committed to develop OPEN LOOK applications.

With the future of OPEN LOOK decided, USL is now working on that future. OPEN LOOK will follow the path determined in UNIX International and in standards organisations such as X/Open and IEEE P1201.

USL is now developing internationalisation capabilities in OPEN LOOK. We are doing this in conjunction with UNIX International companies, such as Fujitsu. A prototype of a future OPEN LOOK internationalisation product is now being made available to UNIX International companies.

It is straightforward to realise that Graphical User Interfaces will be an imprtant part of the future of the UNIX Operating System. OPEN LOOK is very much a part of that future with UNIX System V.

| • Existing Applications: | |
|---|---|
| Bristol Group (Fax Software) | Ingres (DEMS Interface) |
| Elan (Text Processing) | VersaSoft (dBASE Workalike) |
| EXOC (Prototyping Tools) | Visual Edge (UIMS) |
| Ficor (Business Graphics) | |
| **• Developing Product:** | |
| Advanced Software Automation | (CASE) |
| ADAPTI | (Financial) |
| Answer Computer | (CASE) |
| Arbor Text | (E-pubs) |
| Ashton-Tate | (Database) |
| Auto-Desk | (Arch/Engr) |
| BBN | (Data Analysis) |
| Bristol Group | (Visualisation) |
| Conceptual Structures | (E-pubs) |
| Control Data | (Education) |
| Convergent Solution | (Database) |
| Crosswind Systems | (Office Auto) |
| Cullinet/CA | (Database) |
| Digital Solutions Inc. | (Financial Svsc) |
| Entropic speech | (Signal Processing) |
| Expert Object Corp | (CASE) |
| Gateway Design | (Electronic Design) |
| Georgetown University | (Medical) |
| Graphic Software Systems Inc. | |
| H.T. Graphics | (Financial Services) |
| IDE | (CASE) |
| Information Builders | (Focus - DBMS) |
| Informix | (Database) |
| IXI | (Desktop) |

| | |
|---|---|
| Lanson-Davis | |
| Lotus | (Spreadsheet) |
| Mernex Information | (E-pub) |
| Metcet Research | (CIM) |
| MicroFocus | (Dev Environment) |
| Micronim | (Database) |
| Mihalism Associates | (Visualisation) |
| Natural Language | (CASE) |
| Natural Language | (Database) |
| Netlabs Inc. | (Networking/Comm) |
| Numerical Design | (Visulisation) |
| Oracle Corp | (Database) |
| P-Stat | (Data Analysis) |
| Quintus | (AI) |
| Reasoning Systems | (CASE) |
| Research Systems Inc. | (Visulisation) |
| Saber Software | (CASE) |
| SAS | (Data Analysis |
| SCO Inc. | (Visualisation) |
| Sherrill Lubinski | (Visualisation) |
| SoftQuad | (E-pubs) |
| Softscience | (CASE) |
| Sybase | (Database) |
| System Strategies Ltd | (Interface) |
| UNICAD | (CASE) |
| Unify Corp. | (Database) |
| University of Pennsylvania | (Medical) |
| V-Systems Inc. | (Fax Software) |
| Wolfram Research | (Data Analysis) |
| WordPerfect | (Text Processing) |

Figure 5: OPEN LOOK GUI ISV Commitment

## Name Change

UNIX System Laboratories Europe Ltd, formerly AT&T UNIX Software Operation Europe, has responsibility for the sale and marketing of UNIX System V and associated products throughout Europe. The company is a subsiduary of UNIX System Laboratories Inc, a wholly owned subsiduary of AT&T.

All the UNIX trademarks that were formerly owned by AT&T have been transferred to USL Inc. These include UNIX, TUXEDO, OPEN LOOK, XWIN, Writers Workbench, Instructional Workbench and Documenters Workbench.

They are registered trademarks of the UNIX System Laboratories Inc in the USA and other countries.

# C++ Column

*Mark Rafter*
*rafter@warwick.uucp*

*Warwick University*

Mark Rafter is a lecturer in Computer Science at Warwick University. He has been using C++ in research and teaching since Jan 1985. At present his research interests involve the use of parameterised classes and multiple inheritance to build a class library for use in hyperbolic geometry.

**The Annotated C++ Reference Manual,**
Margaret A. Ellis and Bjarne Stroustrup,
Addison-Wesley,
ISBN 0-201-51459-1,
(UK) Hard cover Price £31.45,
UK publication date: late August 1990

The book starts:

> "This book provides a complete language reference manual for the expert C++ user."

This is accurate!

It is a comprehensive statement of what C++ is, why it is the way it is, why it isn't the way it isn't, how its parts relate to each other, how the language relates to its cousins, how aspects of the language can be implemented, and where the language is heading.

The book is not introductory or tutorial in nature – it contains no complete programs and no explanation of how the various language constructs should be used in producing well structured software. It is not aimed at the novice or the casual C++ programmer. It is a reference manual for language specialists, C++ experts and for serious users of C++ that wish (or need) to obtain a complete grasp of the language.

The book contains the current definition of the C++ language and is the document that ANSI have taken as the starting point for the C++ standardisation work. It has 18 chapters in 408 pages, a 39 page index and is extensively cross-referenced. It is organised as an expanded form of the C++ Language Reference Manual that was distributed with release 2.1 of C++ from AT&T. (This, in turn, is a corrected form of the Draft C++ Reference Manual that was distributed with release 2.0 – which in its turn started life as the Reference Manual chapter (pages 245-311) of Stroustrup's previous book "The C++ Programming Language").

Each section of the AT&T 2.1 C++ Reference Manual expands to become a chapter in the book. These chapters combine three types of material: the reference manual proper, commentary interwoven with this reference material, and major discussions which are placed at the ends of chapters. The typesetting clearly distinguishes commentary from the reference material upon which it is commenting.

The reference material proper is almost exactly the same as the material in the AT&T 2.1 C++ Reference Manual (the differences are discussed below). Taken by itself this material is dry and leaves the interested reader asking many questions of the form "Why does the language allow this rather than that?".

The interwoven commentary answers many of these questions where they naturally occur, sometimes including examples of the consequences of alternative design choices. This will often cause readers to react: "Oh! I hadn't thought of that."

The commentary often presents examples of languages constructs in the form of program fragments together with explanatory text. Among many other things, the commentary is used to emphasise important points, to highlight fine distinctions, to draw the reader's attention to

implications that may be easy to overlook, to clarify points in the reference material that may be thought to be ambiguous or vague, to give examples of things that are not in the language, to discuss the relationship to ANSI C, Classic C and earlier variants of C++, to discuss implementation issues, and occasionally to orient the the less-experienced reader.

Major discussions are placed at the end of thirteen of the chapters. These include discussions of the history of the language, its design philosophy, portability considerations and future languages features. Also covered are implementation techniques for overloaded functions, pointers-to-members and multiple inheritance. Much of this material has been published elsewhere, some of it is of independent interest – almost all of it is necessary for a deep understanding of the language and its spirit.

Templates and Exception Handling are the future language features. They are neither part of the language definition nor of any widely available implementation. Two chapters composed entirely of commentary describe the current proposals in these areas.

Earlier this year I had gone through the 2.0 Draft Reference Manual with a fine tooth comb, so I was familiar with a previous version of the unannotated material. That had been done in a "random access" fashion and had been hard work.

When I picked the up the book I was a bit daunted by its size, but in fact about 50% of it is explanatory commentary. The interweaving of commentary and reference material seems to have kept my attention far better that the raw reference material had done previously. I found that I could quite happily read from the front to back without a lot of page thrashing for forward and backward references. I attribute this in part to the careful location of the commentary which is full of interesting items and relevant examples.

The language itself has been cleaned up in places. For example, initialisers for references must not introduce temporaries so a reference-to-int can no longer be initialised with the constant 1 you must use a reference-to-const-int here instead. This is an important change and it naturally provokes the question: "How does my C++ system relate to the language defined by the book?". Of course this is not a question that the book should answer – each implementation must answer it, preferably by relating it to the language definition reference material in the book.

For AT&T C++ 2.0, the major differences from the book are:

- the introduction of true nested classes,
- deleted arrays no longer specify their length,
- typedefs and enumerations nest within classes,
- initialisers for references must not introduce temporaries,
- a pointer to constant may not be deleted,
- protected base classes are allowed,
- default constructors may have default arguments,
- the increment and decrement operators may have distinct prefix and postfix definitions.

At present there is no implementation of C++ that fully implements the language defined by the book – AT&T C++ 2.1 comes close but does not implement the last three changes. However, the Release Notes that accompany AT&T C++ 2.1 are recommended reading. These give an extremely clear statement of the relationships between AT&T's 1.2, 2.0 & 2.1 languages and implementations, and how migration from 2.0 to 2.1 will be managed.

Unfortunately, the AT&T C++ 2.1 Language Reference Manual is *not quite* identical to the reference material contained in the book – for example see 8.2.2 (arrays of references). In all cases where I checked it was the book that was correct.

I finished the book with a better understanding of the language than I had started with. This is due in part to the cleanup the language has undergone, but it is mainly due to the strengths of the book itself. It is well organised, contains very few typos, and is well produced. The major commentaries gather together important material that has been scattered through the literature and the interwoven commentary transforms the important, but dry, reference material into a hard, but interesting, read. Almost all the target readership will learn something from this book – most will learn a lot.

## STOP PRESS

News has just reached me that the ANSI C++ standardisation committee has voted to accept the changes to C++ enshrined in the "Annotated C++ Reference Manual" into the draft standard. In addition the chapter on templates has also been accepted.

---

## !%@:: A Directory to Electronic Mail Addressing and Networks Second Edition, 1990

The new 1990 edition of !%A:: A Directory of Electronic Mail Addressing and Networks, by Donnalyn Frey and Rick Adams is now available. This new edition provides readers with a directory and usage guide to approximately 130 of the world's research and educational networks, as well as commercial networks. The network information has been updated for 1990, with many new networks added. The directory makes it easy for readers to find networks they can use to reach other people around the world and guides readers in how to use them. It also assists readers in finding someone's email address and sending mail. The book is in an easy-to-use short reference format.

The directory is of use to system administrators who field electronic mail questions, network administrators who work with networks in other countries, researchers who want to get in touch with other researchers, conference attendees with many contacts, and others who routinely send email. Each network section contains general information about the network, as well as address structure and format, connections to other sites or networks, facilities available to users, contact name and address, cross references to other networks, network architecture, future plans, date of the last update, and a map showing the network location. Also included is a three-way index to network name, network type, and country, as well as a list of many of the world's second and third level domains.

This new edition contains:

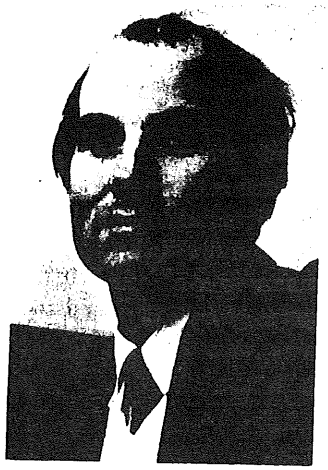
- information on new networks such as AlterNet, CANET, CA*net, EASInet, InterEUnet, IXI, MFENET-II, TUVAKA, XLINK, and YNET
- updated information on networks that are reorganising or have reorganised, such as BIONET, ESNET, MFENET, NYSERnet, and OnTyme
- information on networks in the Soviet Union, Eastern European countries, and the People's Republic of China
- networks not in the first edition, such as ATT Mail, KREOnet, and SCIENCEnet
- updates of most of the existing networks described in the first edition which was published in 1989.

Many networks have had significant service or architecture changes, as well as new network connections or connections to new computers since publication of the first edition. For example, AARNet, the new major Australian network, was in the planning stages when the first edition of the book was published. AARNet is now a functioning network with connections both within Australia and to other countries. As another example, the Canadian national TCP/IP network, CA*net did not exist in 1989. CA*net is beginning operations in 1990 and soon will be the major network in Canada.

Small, but significant changes occurred in many other networks, changes such as new higher-speed lines, new connections in the country, and more. For example, mcvax, the major central networking computer for EUnet in Europe, was renamed mcsun. In many cases, network administrators had changed, so the authors provide new contacts. The authors include the new connections to the Soviet Union, Eastern Europe, and the People's Republic of China. Many of the connections existed before for private use. Now, the connections are open and people are encouraged to contact colleagues in these countries.

This new edition is the most up-to-date guide for directing your electronic mail; it is a real time saver. The book will continue to be updated every ten to twelve months. Readers who fill out the response card in the book have the option of either receiving notification of updates or receiving the updated edition automatically at a 25% discount.

# Report on ISO/IEC JTC1/SC22/WG15 (POSIX)



*Dominic Dunlop*
*domo@tsa.co.uk*

*The Standard Answer Ltd*

Dominic Dunlop, an inveterate busy-body who likes to see order in all things – except on his desk-top – has been making a nuisiance of himself in UNIX standardization forums since the early nineteen-eighties, when he got tired of the needless differences between the many systems that were targets for his software porting activities. He would have gone to the recent UKUUG conference in London, and to the POSIX meeting in Danvers, but was busy getting married at the time...

## Report on ISO/IEC JTC1/SC22/WG15 (POSIX) Meeting of 11th – 15th June, 1990, Paris, France

### Introduction

Working Group 15 of Subcommittee 22 of Joint Technical Committee 1 of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC22/WG15), or, more briefly, the ISO POSIX working group, met in Paris, France, from the 12th to the 15th of June. The meeting was hosted by AFNOR, (Association française de normalisation), the French national standards body, at its offices in La Défense, a high-rise business district a brisk train-ride away from the city centre, and was preceded on 11th June and the morning of the 12th by meetings of the rapporteur groups on conformance testing, internationalization and security. Attendance was good, with thirty "experts" (as the *ISO Directives* style us) representing nine countries, plus the European Community.

I was present at the main meeting and at the internationalization rapporteur group as an observer with the brief of reporting back to you. This report is the fourth jointly commissioned by the European UNIX systems User Group (EUUG) and USENIX. As usual, it's a little imprecise in its references to ISO. Strictly, most of these should be to JTC1, or to some part of JTC1. Where precision is needed, I use it and give an explanation, but for the most part I refer simply to ISO, so as to avoid getting bogged down in unnecessary detail. If you have any comments, or need clarification or further information, please contact me at the mail address above.

First, a summary of the most important aspects of the meeting:

### Summary

- POSIX.1, the operating system interface standard, should be published as international standard 9945-1 Real Soon Now. As I write, the ballot on the document has yet to close, but it seems unlikely that any of the twenty countries voting will oppose acceptance. The meeting identified a number of trivial editorial changes to the current draft international standard, and these, taken together with continuing nit-picking comments from ISO's central secretariat, should result in a document which ISO will print. Its technical content will be very close to that of

ANSI/IEEE Std. 1003.1:1988, so implementors following the U.S. standard can be assured of ISO compliance when 9945-1 finally sees the light of day.

- POSIX.2, the shell and tools standard, faces a bumpier ride before becoming international standard 9945-2. In an ISO ballot on acceptance of draft 9 of IEEE 1003.2 as a draft international standard, six countries voted against, with just five in favour. This is more of an embarrassment than a set-back: hindsight suggests that it was just too early to hold a ballot.

- Showing its increasing concern and frustration at the lack of apparent progress within the IEEE on a (programming) language-independent version of the POSIX operating system interface standard, WG15 has refused to "register" the current, largely language-independent, draft of the 1003.4 realtime extensions standard on the grounds that it makes little sense to have language-independent extensions to a base standard which is specified in terms of the C language. Similarly, the group failed to register 1003.5 (Ada binding) and 1003.9 (FORTRAN-77 binding) because POSIX.1 lacks an explicit service definition to which they can bind.

- The failure to register these documents causes a hiccup — albeit a discreet one — in the synchronization between IEEE and ISO POSIX standardization schedules. In the hope of avoiding such situations in the future, an informal "speak now, or forever hold your peace" mechanism has been put in place, allowing the international community to comment on the subject area, format, presentation and approach of IEEE documents at an early stage in their preparation.

- Had it not been for this upset, the working group would have adopted a firm synchronization plan so as to ensure that the work of IEEE and of ISO is closely coordinated in the future. As it is, the "U.S. member body" — ANSI — has been asked to provide a revised plan for the working group's October meeting in Seattle.

- The Open Software Foundation, UNIX International and X/Open are cooperating on a common approach to conformance testing, known as Phœnix. Governmental organizations with a strong interest in the field, such as the National Institute for Science and Technology (NIST) and the Commission of European Communities (CEC), seem broadly to welcome this move — even if the unaccustomed show of tripartite unity is, as one security rapporteur put it, "a bit spooky".

- At an evening reception hosted by AFUU (Association française des utilizateurs d'UNIX), the French UNIX users' group, Mike Lambert of X/Open said that "our hand is extended" to the international standardization community, with which his organization hopes to work in finding some more timely and responsive way of delivering formal consensus standards for open systems. The definition of this mechanism is left as an exercise for the reader — or for the international standards community. Perhaps Mike has come to realize that standardisers too are prone to the Not Invented Here syndrome, and so must believe that they thought of something themselves before they can accept it.

- Just to keep us all on our toes, ISO has updated its Directives, with the result that the procedure for turning a base document — such as one of the IEEE drafts — into an international standard is subtly changed. We now have to forget about Draft Proposals, and turn our minds instead to Working Drafts and Committee Drafts. Sigh...

The rest of this report gives more detail most of these topics.

## 9945-1 — Operating System Interface

You may recall from my report of WG15's last meeting in October, 1989, that the group had in effect told ISO's central secretariat that, while the then-current draft of IS 9945-1 was not perfect, it was, in the group's opinion, good enough to publish, particularly since we'd undertake to fix things up on short order, allowing an improved version to be published within a year of the first edition.

This proposal did not fly. Firstly, the central secretariat (now dynamically known as ITTF, the Information Technology Task Force), refused to publish a document that looked much more like an IEEE standard than an ISO standard; secondly,

they deemed the changes needed to polish up the draft to be sufficiently great that it should go out to ballot again in order to provide a formal check that it was still acceptable to the group. This was duly done; the ballot closed on 29th June, and is expected to pass very comfortably.

Nevertheless, the ballot gave people the opportunity to comment formally on the document again, with the result that a number of small bug-fixes and clarifications were suggested, and were accepted for incorporation into the draft. We judge — and we hope that ITTF agrees — the changes are strictly editorial, and so will not merit yet another ballot. ITTF, which, in discussion with the IEEE, had slightly bent its drafting and presentation rules so as to come up with a compromise satisfactory to both parties, also suggested further changes, some in areas that we thought had already been addressed. This is a cause for concern: the prospect of the standard never being published because its layout and language do not meet some ill-defined guidelines does not appeal. Consequently, we are seeking "written harmonised editorial requirements" from the IEEE and ITTF.

The ballot also produced a number of suggestions in the area of internationalization, such as how to handle (and indeed, how to refer to) wide, or multi-byte, characters. To have acted on these comments would have changed the technical content of the draft standard — the equivalent of sliding down a snake in the game that leads to eventual publication. Happily, those who suggested the changes were content to leave these issues for the second edition of the standard, rather than further delay the first edition.

The single exception that we could get away with was to replace Annex E's[1] example international profile for the hypothetical (and extremely odd) land of Poz with a real example for the (only slightly odd) country of Denmark. Although this is a large change, it can be made because the annex (ISO-speak for appendix) is "non-normative"; that is, it is an explanatory example rather than a part of the official standard.

Messaging, an issue which is currently exercising the minds of those concerned with the next edition of the 1003.1 standard[1], was also passed over by WG15: nobody had a strong preference for either the X/Open proposal or the UniForum proposal, so 1003.1 will have to handle the issue without guidance from from the ISO working group.

Where all does this leave us? With no published international standard as yet, but with a very good prospect of having one this year. Until it arrives, keep using the bilious green book, IEEE std. 1003.1:1988[2], as its technical content is very close to that of the eventual ISO standard[3].

## 9945-2 — Shell and Tools

Earlier in the year, there had been a ballot on moving forward draft proposal (DP) 9945-2, *Shell and utility application interface for computer operating system environments*, to become a draft international standard (DIS). Basically, while a DP is allowed — even expected — to differ considerably from the international standard which ultimately results, a DIS is expected to be in a format and to have contents which are very close to those of the ultimate standard[4]. That the ballot was six against to just five for (with nine countries not voting) indicates that the consensus is that 9945-2 has to go through quite a few more changes before it can be acceptable as an international standard.

Many of these changes concern internationalization, as this topic affects POSIX.2 considerably more than POSIX.1. For example, the example Danish national profile to be incorporated into 9945-1 is just a part of the work that DS (Dansk Standardiseringråd) has done on the topic; the rest affects 9945-2. There is also an unresolved issue concerning the definition of collation sequences (sorting orders) for

1. This material is not in the published IEEE std. 1003.1:1988.

2. You can buy a copy by calling IEEE customer service on +1 908 981 1393 (1 800 678 IEEE inside the U.S.) and giving them a credit card number. The cost is $37, plus $4 for overseas surface mail, plus another $15 for airmail. Alternatively, your national standards body may be able to sell you a copy. For example, BSI in the U.K. has them for sale at £24.

3. A new edition of ANSI/IEEE 1003.1, to be published this year, will be identical in technical content to the ISO standard.

4. DP 9945-2 is the last that we will see; under the new directives, DPs are no more; they are replaced by working drafts (WDs), which become committee drafts (CDs) before becoming DISs. This is not a big deal.

international character sets. Utilities such as *sort* clearly need to know about collation sequence, and so do the regular expression-handling utilities and functions defined by POSIX.2. The problem is that nobody in ISO seems to want to handle the matter. In particular, JTC1/SC2, which standardises coded character sets, sees its job as assigning codes to characters, not as saying anything about how those codes should sort[5]. This is a reasonable attitude: collation is a surprisingly complex field, and to attempt to cover it in coded character set standards would break the ISO rule of one topic, one standard. This is all very well, but 9945-2 needs somebody to do the work, and WG15 may end up doing it itself if pleas for help from elsewhere in ISO fail.

More work is on the way: 1003.2a, the User Portability Extension to POSIX.2, was registered for ballot as a Proposed Draft Amendment (PDAM) to DP 9945-2, giving the international community a chance to say what it thinks of the idea of standardizing interactive commands such as *vi* and language processors like *cc* — or rather *c89*.

## Coordination

The coordination arrangements which will make IEEE and ISO work on POSIX march forward in lock-step were almost complete before the small international rebellion on the matter of language independence made us stumble. (See below.) Because WG15 failed to register 1003.4, 1003.5 and 1003.9 at this meeting, the plan must be adjusted, although little slippage should result. We'll try to jump on board the revised plan at the next meeting.

## Internationalization

In the one and a half days before the main meeting of WG15, its three rapporteur groups met. I attended the internationalization meeting, which had a hectic time of it: as the only rapporteur group directly concerned with the current content of 9945-1 and -2, we had a lot of comments to sift through prior to the main meeting. This we did, by the skin of our teeth. Our conclusions are largely reflected in the actions on the two drafts, reported above.

## Security

The security rapporteur group is just getting off the ground. As with internationalization, activities scattered throughout JTC1 have to do with security. But in contrast to the current situation for internationalization, for security there is a (very new) subcommittee, SC27. Conceivably, SC27 could define some all-encompassing ISO security model[6] which would not suit POSIX and the work of 1003.6, which is eventually to be integrated into the 9945 standards. The rapporteur group is doing all that it can to prevent this from happening. Happily, ISO is already aware of the issue, and has formed a special working group on security, to which WG15 will be sending a representative.

## Conformance Testing

The proceedings of the rapporteur group on conformance testing were rather overshadowed by the announcement of Project Phœnix. Quite from what ashes this has risen we did not learn; however, as it involves cooperation between the Open Software Foundation (OSF), UNIX International and X/Open, it is difficult to resist the temptation to speculate. But resist I shall...

The goal of Phœnix is to develop a common conformance testing suite and methodology for the three organizations, or, to put it another way, to harmonize their activities in this area. Harmonization of standards for open systems is an important issue for WG15 in general. The issue affects the rapporteur group on conformance testing in particular because the European Community and NIST are giving a high priority to accrediting test houses which can check conformance to open systems standards. This means that they need to ensure that uniform test methods are being consistently applied. The rapporteur group will address this issue at its next meeting. In the mean time, WG15 has registered the current draft of the first part of 1003.3, which describes general test procedures, and we will

5. For oblique confirmation from "the father of ASCII", see [2].

6. ISO likes models, and they're not without their uses. Work on a useful model for open systems is under way in several forums.

review it before our next meeting — despite the fact that there is as yet no well-defined route by which POSIX.3 can become an international standard.

## Language Independence

The issue of a making the POSIX standards independent of any particular computer language came to the fore at this meeting. What's all the fuss about? Well, ISO has firm — and, in my view, sensible — ideas about how to write standards which define the services available from information processing systems. Building on the doctrine that one standard should address just one topic, there should be, says ISO, one document defining the service, and one or more documents describing ways of accessing that service. For example, a browse through the open systems interconnection standards reveals several groupings such as IS 8072, *Transport Service Definition* and IS 8073, *Connection oriented transport protocol specification*; and IS 8649, *Service definition for the Association Control Service Element*, and IS 8650, *Protocol specification for the Association Control Service Element*[7]. Similarly, in text communication, there is IS 9066-1, *Reliable transfer — model and service definition* and IS 9066-2, *Reliable transfer — protocol specification*, and in graphics, IS 7942, *Graphical Kernel System functional description* and IS 8651-1 through -3 giving GKS language bindings for FORTRAN, Pascal and Ada. (8651-4, giving bindings for C, is in the works.)

POSIX, ISO has ordained, must eventually go along with this model, with the result that IS 9945-1, -2, and -3 (Operating system interface, shell and utilities, and administration respectively) will be concerned simply with defining services, and will not be bound to any particular programming language. The key word here is "eventually": because of the pressing need for an international standard for POSIX, a waiver has been granted, allowing the first version of the 9945-1 and -2 standards to be a combination of (purists might say "a collision between") a service definition and a C language binding to those services. The expectation is that a future revised new edition of each standard will be language-independent, and that bindings for the C language will be published as a separate standard at the same time[8].

So far, so good. But this is where the problems start. While this mandated future appears a long way off if one looks at the IEEE's work on POSIX.1, it seems very close when POSIX.4 (realtime extensions), POSIX.5 (Ada bindings) and POSIX.9 (FORTRAN-77 bindings) are considered. In the case of POSIX.4, language-independent extensions are being proposed for a standard which is not itself (yet) language-independent. And POSIX.5 and POSIX.9 define bindings to a set of services which is not explicitly defined, but rather is defined implicitly in terms of the binding to the C language given in POSIX.1. In the absence of a clear service definition, it is no surprise that, for good reasons which have to do with their respective languages, the Ada, C and FORTRAN versions of the operating system interface appear currently to be binding to slightly different sets of services.

To some, the whole issue of language independence seems like an unnecessary and irksome imposition by ISO. In a recent posting to comp.std.unix[3], Doug Gwyn wrote:

> [Those of us who worked on 1003.1] did NOT set out to create a language-independent standard; C was specifically chosen for the obvious reason that it was the SOLE appropriate language for systems-level programming on UNIX, for a variety of reasons, including the fact that the UNIX kernel has a marked preference for being fed C data types.

It is certainly true that, because, back in 1985, all the base documents for the IEEE POSIX work were written in terms of a C language interface, the working group made a good pragmatic decision to

7. Browsing through OSI standards may be a cure for insomnia. On the other hand, it may aggravate hypertension...

8. Under ISO's procedures, the bindings standards for POSIX will be allocated an ISO standard number just as soon as we register a base document for one of them. Until that happens, I shall have to refer to "future bindings standards", rather than having a convenient number to use as a handle.

produce a standard centered on C. A different decision would have resulted in the standard which took longer to get out of the door, and which would not have been any more useful to its primary audience — committed UNIX users concerned with the divergence among implementations of their chosen operating system. But the success of UNIX and of its offspring, POSIX, has greatly widened the audience for the standard. Whether we like it or not, ISO has revisited the original decision so as to ensure that the international standards for POSIX meet the needs of this new audience. As a result (to continue quoting from [3]):

> This "language binding" nonsense was foisted off on P1003 in an attempt to meet ISO guidelines. I think it must have been adopted by ISO as the result of Pascal types insisting that they never have to use any other language.

Countering this, I would contend that, while the number of "Pascal types" is too small for their opinion to be of prime concern, the number of FORTRAN types, COBOL types and perhaps even of Ada types is large enough that it would be at least polite to provide some well-defined means whereby these communities can create bindings which allow them to hook into POSIX services without having to learn a new programming language. In the future, the growing C++ community may decide to define the interface to POSIX services in an object-oriented manner; Steve Carter paid us a flying visit with news from the ANSI X3J16 C++ committee in order to open up channels of communication.

Consider another topic which has come to the fore as POSIX has moved into the international arena: internationalization — mechanisms which will allow non-English speakers to use POSIX-compliant systems without having to learn a new natural language. Like the current movement towards separating service definitions from bindings, this involves a considerable amount of work, yet does not appear to provide much that is of use to UNIX' original community of technical users. Accommodating the preferences (prejudices?) of ever greater numbers of people is, it seems to me, part of the price of success for the UNIX operating system. And it may well pay dividends. For example, internationalization work on regular expressions and collating has resulted in facilities which will be of use even to English speakers.

Returning to the matter of the programming language used for bindings, it is true that AT&T-derived UNIX implementations prefer a diet of C data types. However, it certainly was an aim of 1003.1 to allow hosted POSIX implementations, which might well be riding on underlying operating systems with entirely different tastes. As a topical example, lightweight kernels such as Chorus and Mach live on messages, suggesting that their services could be bound to a data stream encoding[9]. I suspect that anybody who has tried to make *ioctl*( ) work across a network wishes that UNIX had anticipated their needs by following such a model from the start. But it didn't, and to redefine it in these terms would be a large piece of work which (thankfully) is a long way beyond the scope of WG15.

There is no way in which all such requirements could have been anticipated, and accommodating the most important of them as the need arises inevitably causes pain. Both language independence and internationalization are unanticipated requirements which the international community wants retrofitted to POSIX on short order. And it's ANSI, as provider of the base documents to ISO, and the IEEE, as the body accredited by ANSI to produce the documents, that get beat on to do the real work, and to suffer the pain.

In the view of WG15, the real work needed to make POSIX.1 a logical base for extensions such as POSIX.4, POSIX.5 and POSIX.9 is not being done fast enough. Trouble is, all standards are produced by volunteers — often volunteers who have had to make a case to their managements that there's some percentage in their company being involved in standards work. There is clearly an eventual percentage in language independence for suppliers of POSIX-conformant systems if it encourages users of languages not traditionally found on UNIX systems to migrate to POSIX. But sadly, while not in any way

9. More ISO-speak: broadly, if you have a protocol that lives above layer five (session) of the OSI stack, you'd better call it a data stream encoding. For example, the protocol for the X Window System™ is a data stream encoding by this definition.

criticizing the quality of the work done to date, there aren't enough IEEE volunteers interested in recasting POSIX.1 into language-independent form.

Maybe, just maybe, if the international community is more interested than the U.S. in getting this work done, WG15 should encourage more people from outside the U.S. to participate actively and directly in the work of the IEEE. (Or, to put it another way, encourage more organizations outside the U.S. to put their hands more deeply into their pockets in order to pay for people to attend IEEE POSIX working group meetings.) The alternative is that WG15 does the work itself — an alternative I'd rather not contemplate.

For now, two action items on ANSI from WG15 sum up the situation:

> Pursue with vigour the production of a language-independent version of both 9945-1 and P1003.4 in conjunction with a C language binding for each in order that they are eligible as replacements for 9945-1:1990.
>
> Request the IEEE to expedite the completion of the language independent specification of 9945-1 that is precisely functionally equivalent to the existing 9945-1:1990 and provide a C language binding that is syntactically and semantically identical; and request that a detailed proposal status report on this issue including a synchronization proposal be presented at the next meeting of WG15.

## Next Meeting

The next meeting of WG15 is in Seattle from 23rd to 26th October — the week after the IEEE POSIX working group meeting in the same city (and the same week as the EUUG meeting in Nice, France[10]). Should be interesting!

## References

1. June, 1990 Standards Update, Jeffrey S. Haemer, comp.std.unix Volume 20, Number 66, USENIX, 30 June 1990
2. Letter from R. W. Bremer, pp 34-35, *Byte*, volume 15, number 6, June 1990
3. Doug Gwyn, comp.std.unix Volume 20, Number 51, USENET, 27 June 1990

10. In two meetings, WG15 has managed to clash both with summer USENIX and with autumn EUUG. It almost looks as if we do it on purpose! But we don't, and will try to do better next year...

# Puzzle Corner

*Mick Farmer*
*mick@cs.bbk.ac.uk*

*Birkbeck College*

Mick is a lecturer at Birkbeck College (University of London) and the Secretary of the UKUUG. His interest is in all aspects of Distance Learning and he is the Senior Consultant (Software) for LIVE-NET, an interactive video network connecting London's colleges. He is also a member of the University's VLSI Consortium, mainly because the design tools draw such pretty pictures.

Hello peeps,

## Solution to Puzzle Number 12

The force pressing the tile against the roof is 2 × cos 20° and, as the coefficient of friction is 0.5, the total frictional force at incipient motion is 0.5 × 2 × cos 20°, or 0.9397 lbs. The force causing sliding is 2 × sin 20°, or 0.6840 lbs.

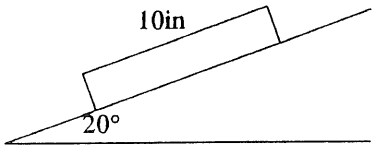



Figure 1 – The Creeping Roof Tile

When thermal expansion causes motion with respect to the roof, the total frictional resistance of 0.9397 lbs. must be developed, yet in general the tile will not move so that the overall force is 0.6840 lbs. These conditions are met if the difference (0.2557 lbs.) occurs half in one direction and half in the other. Thus a frictional resistance tending to shove the tile down the roof of 0.12785 lbs. is developed on the heating cycle by the part of the tile above the centre of expansion. This is opposed by a frictional force of 0.81185 lbs. (0.6840 + 0.12785), developed on heating, by the part of the tile below the centre of expansion, tending to hold the tile in place. The centre of expansion is 1.361" (0.12785 / 0.9397 × 10) below the top of the tile. On expansion the centre of the tile moves down 0.00109" ((5 - 1.361) × 6 × $10^{-6}$ × 50). On contraction, the centre of contraction is 1.361" from the lower end of the tile and the centre creeps down an additional 0.00109" making the daily movement 0.00218".

## Solution to Puzzle Number 13

To fully utilise the toaster, both sides must do an equal share of the work. Three sides of the bread slices must be toasted on each side of the toaster:

| | |
|---|---|
| 0.00 - 0.05 | Put A slice in left |
| 0.05 - 0.10 | Put B slice in right |
| 0.55 - 0.57 | Turn slice A |
| 0.60 - 0.65 | Remove slice B |
| 0.65 - 0.70 | Put slice C in right |
| 1.07 - 1.12 | Remove slice A to plate (done) |
| 1.12 - 1.17 | Put slice B (second side) in left |
| 1.20 - 1.22 | Turn slice C |
| 1.67 - 1.72 | Remove slice B (done) |
| 1.72 - 1.77 | Remove slice C (done) |

## Challenging Quickies

For this issue, a slight change in format for the puzzles. You're invited to keep track of the time taken to reach the solution of these three *quickies*. This should not exceed five minutes each, since they are chiefly a test of mental agility in finding the best short-cut to the answer.

### Puzzle Number 14

A 1" cube has its outline constructed out of wire whose resistance is one ohm per inch. What is the resistance between opposite corners of the cube?

### Puzzle Number 15

Suppose I think of a number whch you are to determine by asking me not more than twenty questions, each of which can be answered by only "yes" or "no". What is the largest number I should be permitted to choose so that you may determine it in twenty questions?

### Puzzle Number 16

Why must a house whose rooms each have an even number of doors also have an even number of outside entrance doors?

**Loads-a-puzzles,**

Mick

# Call Doc Strange

*Colston Sanger*
*doc.strange@gid.co.uk*

*GID Ltd*

Three months have passed and Colston Sanger is still a senior consultant/tea boy with GID. He's also still a visiting lecturer in the Faculty of Engineering, Science and Mathematics at Middlesex Polytechnic. So what's new? Nothing really, except that he's been editing a book with Tony Elliman of Brunel: *Open Systems for Europe: towards 1992* (Chapman & Hall, forthcoming).

## We Are Under Attack!

Err...yes, well. Following an incident at a gateway machine near me, it seemed like a good idea to discuss system security. Not at the usual level of 'Here are a thousand Berkeley bugs that a bored undergraduate cracker might be ingenious enough to use to break your system', but at the much more mundane level of the thousand and one silly things that system administrators themselves do that compromise the security of their systems. What I have to say is likely to be no more than plain common sense, and is aimed primarily at people who are relatively new to UNIX running a standalone System V machine with, at most, a dial-up connection to the outside world: people who administer large networks of Sun workstations are likely to find this stuff 'old hat', but not irrelevant.

Let me start at the beginning, with an extract from Captain's Log for Saturday, 16 June: [1]

```
********** qwerty **********
uucp qwerty  (6/16-15:36:44,10868,0) OK (startup)
uucp qwerty  (6/16-15:36:57,10868,0) REMOTE REQUESTED (sixnine!/etc/passwd
--> qwerty!/usr1/bloggs/rje/.sixnine.pw (fred))
fred qwerty  (6/16-15:36:57,10868,0) PERMISSION (DENIED)
uucp qwerty  (6/16-15:37:41,10868,0) REMOTE REQUESTED (sixnine!/usr/lib/
uucp/Systems --> qwerty!/usr1/bloggs/rje/.sixnine.sys (fred))
fred qwerty  (6/16-15:37:41,10868,0) PERMISSION (DENIED)
uucp qwerty  (6/16-15:37:56,10868,0) OK (conversation complete tty11 76)
```

I picked this up in the evening when mail messages generated by the standard System V UUCP admin daemons (specifically, `/usr/lib/uucp/uudemon.admin` and `/usr/lib/uucp/uudemon.cleanu` run from root's `crontab` file) showed up in my mailbox. Parenthetically, I also noticed that the gateway's modems were rather busy that evening.

Now, I know who 'fred' really is and, as it happens, I had asked him the day before to send me a couple of files from his machine, which is hidden behind the gateway. So my first response, in all innocence, was to send him a somewhat laconic mail message:

1. In what follows, all names have been changed to protect guilty parties.

```
From colston Sat Jun 16 23:24:10 1990
Subject: You don't *honestly* think...???
To: fred@qwerty
Date: Sat, 16 Jun 90 23:24:10 BST

Fred,

Why do you want sixnine's passwd and /usr/lib/uucp/Systems files?
Haven't you updated uipido since the great 071/081 changes?

Colston
```

Minutes later, on second thoughts, I cancelled it.

Next morning, Sunday, I called him at home. No, he hadn't requested any such thing ... At that point, I rang the gateway's system administrator and, to cut a long long story short, he spent most of a sunny Sunday afternoon in the office. As for fred, regardless of whether he had a password before, he certainly had one on Monday morning.

The incident is trivial in itself. The cracker wasn't able to get anything from my system, first because of normal UUCP permissions (in `/usr/lib/uucp/Permissions`) let alone file access permissions and, secondly, because I run AT&T System V Release 3.2 which implements a shadow password scheme. However, what happened to me also happened to all the other systems with which the gateway has links — with, I hasten to add, the same null result. That's not to say no result. Over the next several weeks, but mainly at weekends, the cracker tried repeatedly to gain access (all logged in `/usr/adm/loginlog`), disrupted dial-up communications and generally behaved like a sort of electronic lager lout — not so much your ordinary 'breaking and entering' style of cracking, more like malicious vandalism. Much more important, the incident caused a lot of irritation and embarrassment for the system administrator and the organisation concerned.

So, apart from the fact that at least one cracker out there has also been reading Clifford Stoll's *The Cuckoo's Egg*,[2] what are the lessons to be learned from the incident?

2. London (The Bodley Head), 1989, ISBN: 0 370 31433 6. Strongly recommended and a good read too.

## Using And Choosing Passwords

First, it's not really a good idea to have user logins on a gateway machine. If you can, regard your gateway as a sort of 'fire-door' between the outside world and your operational systems. If you must have user logins for mail-stops or mail forwarding, then make sure they are blocked (with a '*' or some other uncryptable string in the password field in `/etc/passwd`) or have reasonable passwords.

What is a reasonable password? It's certainly not 'fred', 'wumpus', 'sh*thead' (*without* the asterisk), 'susan', 'jennifer' or any of the other common choices.[3] These are all Berkeley passwords, but the System V-mandated 'fred01' is hardly any better.

There are lots of *DON'T*'s in the literature for choosing passwords:

- *don't* use your login name or your first or last name in any form (reversed, capitalised, doubled, *etc*)
- *don't* use your significant other's or child's name

3. See Daniel V.Klein, 'Foiling the Cracker: A Survey of, and Improvements to, Password Security', *UKUUG Summer 1990 Conference Proceedings*, London, 9-13 July 1990, pp 147-54. Klein reports the results of a password cracking exercise on a sample database of 15,000 accounts: '21% (nearly 3000 passwords) were guessed in the first week, and ...in the first 15 minutes of testing 368 (or 2.7%) had been cracked...On an average system with 50 accounts in the `/etc/passwd` file, one could expect the first account to be cracked in under 2 minutes, with 5-15 accounts being cracked by the end of the first day.' He adds: 'Even though the `root` account may not be cracked, all it takes is one account being compromised for a cracker to establish a toehold in a system.'

- *don't* use easily obtainable information about yourself, such as your telephone, car registration or social security number or the name of the street where you live.

*DO*'s, on the other hand, are a little harder to come by. Some good choices for passwords are:

- the first letter of each word from a line of a phrase, song or poem. For example, 'Ten green bottles standing on the wall' becomes '10gbsotw'; and 'You think I'm stupid?' becomes 'Utlams?'; 'On the first day of Christmas' becomes 'Otldoc' — but you can make them up just as well as I can.
- two short words concatenated with one or more punctuation marks. For example, 'mad+beef' or 'Me?Mug?'.
- alternate between one or two consonants and vowels plus punctuation marks or numbers to make pronouncable, nonsense words. For example, 'ekapitog++' or 'zatoich!'.

Note that just because a password must have at least six characters, that doesn't mean it can *only* have six characters. Although only the first eight characters are significant, the longer a good password is, the harder it is to crack.

Whether you use password aging is up to you (in UNIX/386 it is on by default). Personally, I would encourage people to change their passwords from time to time, but not actually enforce it with password aging. Being faced with '`Your password has expired. Choose a new one.`' while still only half awake doesn't strike me as being conducive to choosing a good password.

One final point: in a commercial environment, the object of attack is as likely to be the financial director as it is `root`.

### Physical Security

In System V, machines come up to the `initdefault` state, defined in `/etc/inittab` as either `2` (multi-user) or `3` (Remote File Sharing) so there is never that window of opportunity of single-user mode on the system console.

Then again, internal crackers can only login directly as `root`, should they discover the password, on the system console: on all other attached terminals they must login as themselves and then `su` to `root` — which means that there will be a record in the `/usr/adm/sulog` file. Even if they are unsuccessful, a record is still written to the file. If they try to `su` to another ordinary user, a record is written to the file. So you can investigate, can't you?

People who leave their terminals logged in while they go off to the pub at lunchtime or, even worse, leave their terminals logged in overnight or *over the weekend* are a menace: they deserve all they get by way of idle or time-out daemons.

As for the external cracker, take care with modem setup: misconfigured modems can fail to logout a user when the line drops — hence the next person dialling in, whoever they may be, can continue with the previous user's login session. (Be especially careful if you are logging in from home and `su`-ing to `root` to do remote system administration!)

### Know Your System

I've talked in a previous column about a system administrator's basic toolkit:[4] half a dozen commands — `who -u`, `/etc/whodo`, `ps -ef` or `ps -efl`, `df` and `sar` — that if you get into the habit of running periodically throughout the day you can use to anticipate and head off problems before they become serious. Even a simple `ls -al` is useful for discovering all sorts of anomalies in the filesystem.

System accounting, sometimes used for charging people extortionate amounts of money, can equally well be employed for getting a feel for what is the 'normal' state of affairs on your machine, as well as for load balancing and detecting minor breaches of company policy such as playing `rogue` or `wanderer` during office hours. More on system accounting later.

One of the irritating things about System V is that there's no one place where log files are kept: you have to look in `/usr/adm` for some,

4. 'UNIX Clinic: No space on integral hard disk drive 0, partition 0', *EUUGN*, Vol.9 No.1 (Spring 1989); pp 63-68.

`/usr/lib/cron` for others, `/usr/spool/uucp/.Admin` for yet others and so on. I'm not sure if it's any better in System V Release 4.0 or in Berkeley UNIX, but at least there you've got `syslogd`. However, it's quite easy to make a simple-minded shell script to keep track of these things, such as the following:

```
# check_log
# Checks various log files for incursions/exceptions
ADMIN=colston
tail /usr/adm/loginlog | mailx -s "Loginlog Check" $ADMIN
tail /usr/adm/sulog | mailx -s "Sulog Check" $ADMIN

# Some of these will be reported by the UUCP daemons anyway
cat /usr/spool/UUCP/.Admin/Foreign | mailx -s "Who this?" $ADMIN
cat /usr/spool/UUCP/.Admin/errors | mailx -s "Trouble with UUCP" $ADMIN
cat /usr/spool/uucp/.Old/Old-Log-1 | mailx -s "UUCP Log" $ADMIN

# For smail - if you're that concerned
tail /usr/spool/uucp/mail.log | mailx -s "Mail.log Check" $ADMIN

# Who's printing what
tail /usr/spool/lp/logs/requests | mailx -s "Print jobs" $ADMIN
```

You'll need to run it out of the `root crontab` at a suitable time each night.

It is also a good idea to arrange for mail to `root`, `adm`, `lp`, `uucp` (and any of the other system logins that get status or error messages by mail) to be forwarded to yourself.

As for the really basic things, you should be aware of the *setuid* and *setgid* executables that are intended to be there. You can find them with:

```
find / -type f \( -perm -4000 -o -perm -2000 \) -print
```

Similarly, you can find files and directories that are writable by others (some shouldn't be) with:

```
find / -perm -2 -print
```

I guess that it would be as well to check the permissions on devices too – both in `/dev` and, if there are any, elsewhere.

### And If You Are Attacked?

Don't fool about pretending to be Europe's answer to Dick Tracy. Enforce user password changes immediately and change your UUCP passwords. Very likely, your UUCP neighbours will also want to change the password that you use when calling them. If you are faced with persistent denial of service attacks — like the malicious vandalism I spoke of earlier — you may want to consider having the telephone numbers of your modems changed as well.

If the cracker has succeeded in breaking and entering, `acctcom` (part of the system accounting sub-system) invoked without any options can give you a complete action replay of all commands issued on your machine. With options, you can narrow down the search and the sheer bulk of output, as in:

```
acctcom -S 22:00 -u fred
```

where the `-S` option selects processes starting at or after 22:00 hours and the `-u` option is the user.[5]

5. Of course, a more sophisticated cracker than fred might not leave any traces...

Here is some sample output:

```
START AFT: Fri Jul 20 22:00:00 1990
COMMAND                          START     END          REAL     CPU     MEAN
NAME        USER   TTYNAME       TIME      TIME        (SECS)   (SECS) SIZE(K)
who         fred   tty21      22:06:01  22:06:01         0.36     0.19     3.26
who         fred   tty21      22:06:04  22:06:04         0.25     0.16     3.88
df          fred   tty21      22:06:06  22:06:06         0.33     0.22     4.43
ps          fred   tty21      22:06:10  22:06:12         2.52     0.59     1.59
date        fred   tty21      22:06:16  22:06:16         0.22     0.11     5.82
su          fred   tty21      22:06:22  22:06:24         2.88     1.28     0.77
who         fred   tty21      22:08:10  22:08:10         0.33     0.20     4.20
sar         fred   tty21      22:08:15  22:08:17         2.82     0.72     1.69
ls          fred   tty21      22:08:21  22:08:21         0.67     0.37     4.32
cat         fred   tty21      22:08:34  22:08:36         2.95     0.26     2.46
sh          fred   tty21      22:08:50  22:08:50         0.16     0.11    22.00
#uucp       fred   tty21      22:08:50  22:08:51         1.39     0.62     2.29
uustat      fred   tty21      22:09:00  22:09:01         1.30     0.50     2.88
```

At the same time, if the damage is at all serious, you might as well brace yourself for a talk with 'the management'. (I know, nobody enjoys the sight of headless chickens running about, but they are going to find out sooner or later anyway.) Better find your distribution set and latest backup too, in case you have to do a complete reload.

### What Do You Do If `root` Leaves?

What do you do if `root` leaves and you have to run the system yourself until you can find a successor?

First, don't start treating him or her as a potential criminal. After all, you are going to need help to understand how your system works. Before he or she leaves:

- go through `/etc/passwd` together so that you know who each login is
- similarly, go through `/usr/lib/uucp/Systems` so that you know and understand why you need dial-up links with each of the systems listed there
- have him or her explain how any customised, locally developed or third-party bits and pieces work from a system administration point of view.

After he or she leaves:

- change the `root` and other system passwords
- arrange to change UUCP passwords.

## Security Is Your Responsibility

Let me end with a quotation from *Computer Weekly*, a UK trade paper, reporting on the first reading of the Computer Misuse Bill currently before the UK Parliament:

Computer owners with unreliable or insecure systems could face compensation claims for damage caused to individuals and an investigation by the Data Protection Registrar.

It goes on to quote the Labour Member of Parliament, Mr Harry Cohen:

"If we accept the premise implicit in the ... Computer Misuse Bill that computers have a special role in the running of our society and it is an offence to misuse a computer, we should also accept that a computer owner must have an obligation to take proper computer security precautions."

## Bedtime Reading

The classic papers by Dennis Ritchie, 'On the Security of UNIX' and Robert Morris and Ken Thompson, 'Password Security: A Case History', often reprinted as part of the supplementary UNIX System documentation, are still well worth reading; as is F.T.Grampp and R.H.Morris, 'UNIX Operating System Security', *AT&T Bell Laboratories Technical Journal*, Vol.63 No.8 Part 2 (October 1984), pp 1649-1672. Most of the books on system administration have a chapter on security: Rebecca Thomas and Rik Farrow, *UNIX Administration Guide for System V*, Englewood Cliffs, NJ (Prentice Hall), 1989, ISBN 0-13-

942889-5 is as good as any and also has a useful chapter on system accounting; and Evi Nemeth, Garth Snyder and Scott Seebass, *UNIX System Administration Handbook*, Englewood Cliffs, NJ (Prentice Hall), 1989, ISBN 0-13-933441-6 covers Berkeley variants as well. *The* book on security is Patrick Wood and Stephen Kochan, *UNIX System Security*, Hasbrouck Heights, NJ (Hayden), 1985, ISBN 0-8104-6267-2 — only available in hardback and expensive, but worth it.

---

**Unix for Users,**
C D F Miller, R D Boyle, A J Stewart, Blackwell Scientific Publications, ISBN 0-632-02416 (UK) Price 12.95 UK Pounds, Soft Back, 250 pages including index,

Reviewed by Kelly Dunlop of Parliament Hill Computers Limited kelly@phcomp.co.uk

The blurb on the back of this book says it is intended "chiefly for readers who have not encountered Unix before" and as such it is a reasonably good book.

It starts off with a simple explanation of what is an operating system and what is Unix for users who are not familiar with computers. It then has a short section which basically says there are lots of systems out there that are called *IX and for the purpose of the book most of them can be classed as Unix. They explain they will try and point out where differences occur in the different varieties of Unix but this is not an easy thing to do and they do fall down in a few places. I like the comment "There are also several claimants to the title of Unix-like system which are, frankly, too far removed from Unix to deserve the name."

There is a section which describes "special characters" and this includes the delete character. It is a common mistake among new users to assume because the delete character is backspace on one machine it will be the same on every other one. It is nice to see a book explain that it is simply a setup feature and it may be different on each machine. The book then progresses to logging into Unix.

It then describes some simple commands to get started using Unix. These include cat, ls, rm, cp and mv. It also tells you how to help yourself on a Unix system (if someone has installed the on-line manual pages of course) by using the man command.

I would have thought pwd and cd belonged here but they are saved until the chapter describing files and directories. This is where they belong but maybe the order of chapters should have been such that the files and directories were explained first.

Now we move on to editing files and a chapter explaining the difference between ex and vi. It describes quite well how to go about creating a file using vi and how to edit it using example files. The command summaries are laid out in tables which makes them easy to find at a later date.

Next comes the chapter about Files, Directories and Users which describes directories, the tree structure of the Unix file system, protection modes and the super-user. This is well laid out and goes through things in an ordered manner.

There are a couple of fairly lengthy chapters describing the use of the Bourne Shell and C Shell as command and programming languages followed by a chapter on more advanced commands. The information contained in then could be found in the Unix manual set but is much easier to digest in this form

The rest of the book describes topics which I feel are more advanced and would be better tackled after the user has gained some experience with the commands he has already learnt.

Included here are C and the Unix Interface, Administration and Maintenance, Text Processing, Other Software, and Communincations and Networking. What concerns me is that a new user will pick up this book, race through it and assume he knows all there is to know about Unix.

Having said that it is one of the better books I have seen. The only criticism is that maybe it goes too far, too fast. It would be useful in an organisation where the "real" users have no time to train new users (common in many places), as a first step onto the Unix ladder.

# AUUGN Back Issues

Here are the details of back issues of which we still hold copies. All prices are in Australian dollars and include surface mail within Australia. For overseas surface mail add $2 per copy and for overseas airmail add $10 per copy.

Back issues are available only to current members or subscribers.

| | | | |
|---|---|---|---|
| pre 1984 | Vol 1-4 | various | $5 per copy |
| 1984 | Vol 5 | Nos. 2, 3, 5, 6 | $5 per copy |
| | | Nos. 1,4 | unavailable |
| 1985 | Vol 6 | Nos. 2, 3, 4, 5, 6 | $5 per copy |
| | | No. 1 | unavailable |
| 1986 | Vol 7 | Nos. 1, 4-5, 6 | $5 per copy |
| | | Nos. 2-3 | unavailable |
| | | (Note 2-3 and 4-5 are combined issues) | |
| 1987 | Vol 8 | Nos. 1-4 | unavailable |
| | | Nos. 5, 6 | $5 per copy |
| 1988 | Vol 9 | Nos. 1, 2, 3 | $5 per copy |
| | | Nos. 4, 5, 6 | $5 per copy |
| 1989 | Vol 10 | Nos. 1-6 | $5 per copy |
| 1990 | Vol 11 | Nos. 1-4 | $5 per copy |

Please note that we do not accept purchase orders for back issues except from Institutional members. Orders enclosing payment in Australian dollars should be sent to:

AUUG Inc.
Back Issues Department
PO Box 366
Kensington NSW
Australia 2033

# AUUG Membership Categories

Once again a reminder for all "members" of AUUG to check that you are, in fact, a member, and that you still will be for the next two months.

There are 4 membership types, plus a newsletter subscription, any of which might be just right for you.

The membership categories are:

Institutional Member

Ordinary Member

Student Member

Honorary Life Member

Institutional memberships are primarily intended for university departments, companies, etc. This is a voting membership (one vote), which receives two copies of the newsletter. Institutional members can also delegate 2 representatives to attend AUUG meetings at members rates. AUUG is also keeping track of the licence status of institutional members. If, at some future date, we are able to offer a software tape distribution service, this would be available only to institutional members, whose relevant licences can be verified.

If your institution is not an institutional member, isn't it about time it became one? Ordinary memberships are for individuals. This is also a voting membership (one vote), which receives a single copy of the newsletter. A primary difference from Institutional Membership is that the benefits of Ordinary Membership apply to the named member only. That is, only the member can obtain discounts an attendance at AUUG meetings, etc. Sending a representative isn't permitted.

Are you an AUUG member?

Student Memberships are for full time students at recognised academic institutions. This is a non voting membership which receives a single copy of the newsletter. Otherwise the benefits are as for Ordinary Members.

Honorary Life Membership is not a membership you can apply for, you must be elected to it. What's more, you must have been a member for at least 5 years before being elected.

It's also possible to subscribe to the newsletter without being an AUUG member. This saves you nothing financially, that is, the subscription price is greater than the membership dues. However, it might be appropriate for libraries, etc, which simply want copies of AUUGN to help fill their shelves, and have no actual interest in the contents, or the association.

Subscriptions are also available to members who have a need for more copies of AUUGN than their membership provides.

To find out if you are currently really an AUUG member, examine the mailing label of this AUUGN. In the lower right corner you will find information about your current membership status. The first letter is your membership type code, N for regular members, S for students, and I for institutions. Then follows your membership expiration date, in the format exp=MM/YY. The remaining information is for internal use.

Check that your membership isn't about to expire (or worse, hasn't expired already). Ask your colleagues if they received this issue of AUUGN, tell them that if not, it probably means that their membership has lapsed, or perhaps, they were never a member at all! Feel free to copy the membership forms, give one to everyone that you know.

If you want to join AUUG, or renew your membership, you will find forms in this issue of AUUGN. Send the appropriate form (with remittance) to the address indicated on it, and your membership will (re)commence.

As a service to members, AUUG has arranged to accept payments via credit card. You can use your Bankcard (within Australia only), or your Visa or Mastercard by simply completing the authorisation on the application form.

# AUUG Incorporated

## Application for Institutional Membership

## Australian UNIX* systems Users' Group.

*UNIX is a registered trademark of AT&T in the USA and other countries.

To apply for institutional membership of the AUUG, complete this form, and return it with payment in Australian Dollars, or credit card authorisation, to:

AUUG Membership Secretary
PO Box 366
Kensington NSW 2033
Australia

• Foreign applicants please send a bank draft drawn on an Australian bank, or credit card authorisation, and remember to select either surface or air mail.

---

This form is valid only until 31st May, 1991

..... does hereby apply for

☐ New/Renewal* Institutional Membership of AUUG $325.00

☐ International Surface Mail $ 40.00

☐ International Air Mail $120.00

Total remitted AUD$_____
(cheque, money order, credit card)

* Delete one.

I/We agree that this membership will be subject to the rules and by-laws of the AUUG as in force from time to time, and that this membership will run for 12 consecutive months commencing on the first day of the month following that during which this application is processed.

I/We understand that I/we will receive two copies of the AUUG newsletter, and may send two representatives to AUUG sponsored events at member rates, though I/we will have only one vote in AUUG elections, and other ballots as required.

Date: __/__/__ Signed: _____

Title: _____

☐ Tick this box if you wish your name & address withheld from mailing lists made available to vendors.

---

*For our mailing database - please type or print clearly*:

Administrative contact, and formal representative:

Name: ..... Phone: ..... (bh)

Address: ..... ..... (ah)

..... Net Address: .....

.....

.....

.....

*Write "Unchanged" if details have not altered and this is a renewal.*

---

Please charge $_____ to my/our ☐ Bankcard ☐ Visa ☐ Mastercard.

Account number: _____ _____ _____ _____. Expiry date: __/__.

Name on card: _____ Signed: _____

---

Office use only: **Please complete the other side.**

*Chq: bank* _____ *bsb* ____-____ *a/c* _____ *#* _____

*Date:* __/__/__ *$* *CC type* ___ *V#* _____

*Who:* _____ *Member#* _____

Please send newsletters to the following addresses:

Name: ..... Phone: ..... (bh)
Address: ..... ..... (ah)
..... Net Address: .....
.....
.....
.....

Name: ..... Phone: ..... (bh)
Address: ..... ..... (ah)
..... Net Address: .....
.....
.....
.....

*Write "unchanged" if this is a renewal, and details are not to be altered.*

---

Please indicate which Unix licences you hold, and include copies of the title and signature pages of each, if these have not been sent previously.

Note: Recent licences usally revoke earlier ones, please indicate only licences which are current, and indicate any which have been revoked since your last membership form was submitted.

Note: Most binary licensees will have a System III or System V (of one variant or another) binary licence, even if the system supplied by your vendor is based upon V7 or 4BSD. There is no such thing as a BSD binary licence, and V7 binary licences were very rare, and expensive.

- ☐ System V.3 source
- ☐ System V.2 source
- ☐ System V source
- ☐ System III source
- ☐ 4.2 or 4.3 BSD source
- ☐ 4.1 BSD source
- ☐ V7 source
- ☐ Other *(Indicate which)* .....

- ☐ System V.3 binary
- ☐ System V.2 binary
- ☐ System V binary
- ☐ System III binary

# AUUG Incorporated

## Application for Ordinary, or Student, Membership Australian UNIX* systems Users' Group.

*UNIX is a registered trademark of AT&T in the USA and other countries

To apply for membership of the AUUG, complete this form, and return it with payment in Australian Dollars, or credit card authorisation, to:

AUUG Membership Secretary
PO Box 366
Kensington NSW 2033
Australia

- Please don't send purchase orders — perhaps your purchasing department will consider this form to be an invoice.
- Foreign applicants please send a bank draft drawn on an Australian bank, or credit card authorisation, and remember to select either surface or air mail.

---

This form is valid only until 31st May, 1991

I, ..... do hereby apply for

| | | |
|---|---|---|
| ☐ Renewal/New* Membership of the AUUG | $78.00 | |
| ☐ Renewal/New* Student Membership | $45.00 | (note certification on other side) |
| ☐ International Surface Mail | $20.00 | |
| ☐ International Air Mail | $60.00 | (note local zone rate available) |

Total remitted AUD$_____
(cheque, money order, credit card)

* Delete one.

I agree that this membership will be subject to the rules and by-laws of the AUUG as in force from time to time, and that this membership will run for 12 consecutive months commencing on the first day of the month following that during which this application is processed.

Date: __/__/__ Signed: _____

☐ Tick this box if you wish your name & address withheld from mailing lists made available to vendors.

---

*For our mailing database - please type or print clearly*:

Name: ..... Phone: ..... (bh)

Address: ..... ..... (ah)

..... Net Address: .....

.....

.....

.....

*Write "Unchanged" if details have not altered and this is a renewal.*

---

Please charge $_____ to my ☐ Bankcard ☐ Visa ☐ Mastercard.

Account number: _____ _____ _____ _____. Expiry date: __/__.

Name on card: _____ Signed: _____

---

Office use only:

*Chq: bank* _____ *bsb* ____-____ *a/c* _____ # _____

*Date:* __/__/__ *$* *CC type* ___ *V#* _____

*Who:* _____ *Member#* _____

Student Member Certification *(to be completed by a member of the academic staff)*

I, ..... certify that

..... *(name)*

is a full time student at ..... *(institution)*

and is expected to graduate approximately ___/___/___.

Title: _____ Signature: _____

# AUUG Incorporated

## Application for Newsletter Subscription

## Australian UNIX* systems Users' Group.

*UNIX is a registered trademark of AT&T in the USA and other countries

Non members who wish to apply for a subscription to the Australian UNIX systems User Group Newsletter, or members who desire additional subscriptions, should complete this form and return it to:

AUUG Membership Secretary
PO Box 366
Kensington NSW 2033
Australia

- Please don't send purchase orders — perhaps your purchasing department will consider this form to be an invoice.
- Foreign applicants please send a bank draft drawn on an Australian bank, or credit card authorisation, and remember to select either surface or air mail.
- Use multiple copies of this form if copies of AUUGN are to be dispatched to differing addresses.

---

This form is valid only until 31st May, 1991

Please *enter / renew* my subscription for the Australian UNIX systems User Group Newsletter, as follows:

Name: ..... Phone: ..... (bh)

Address: ..... ..... (ah)

..... Net Address: .....

.....

..... *Write "Unchanged" if address has*

..... *not altered and this is a renewal.*

For each copy requested, I enclose:

| | |
|---|---|
| ☐ Subscription to AUUGN | $ 90.00 |
| ☐ International Surface Mail | $ 20.00 |
| ☐ International Air Mail | $ 60.00 |

Copies requested (to above address) _____

Total remitted AUD$_____

(cheque, money order, credit card)

☐ Tick this box if you wish your name & address withheld from mailing lists made available to vendors.

---

Please charge $_____ to my ☐ Bankcard ☐ Visa ☐ Mastercard.

Account number: _____ _____ _____ _____. Expiry date: __/__.

Name on card: _____ Signed: _____

---

Office use only:

*Chq: bank* _____ *bsb* _____-_____ *a/c* _____ # _____

*Date:* __/__/__ $ *CC type* ___ *V#* _____

*Who:* _____ *Subscr#* _____

# AUUG

## Notification of Change of Address
## Australian UNIX* systems Users' Group.

*UNIX is a registered trademark of AT&T in the USA and other countries.

If you have changed your mailing address, please complete this form, and return it to:

AUUG Membership Secretary
PO Box 366
Kensington NSW 2033
Australia

Please allow at least 4 weeks for the change of address to take effect.

Old address (or attach a mailing label)

Name: ..... Phone: ..... (bh)

Address: ..... ..... (ah)

..... Net Address: .....

.....

.....

.....

New address (leave unaltered details blank)

Name: ..... Phone: ..... (bh)

Address: ..... ..... (ah)

..... Net Address: .....

.....

.....

.....

---

Office use only:

*Date:* ___/___/___

*Who:* _____ *Memb#* _____