

Stacy

AUUGN

Australian Unix systems

User Group Newsletter

Volume 8

Number 6

The Australian UNIX* systems User Group Newsletter

Volume 8 Number 6

December 1987

CONTENTS

AUUG General Information	3
Editorial	4
President's Report	5
Adelaide UNIX Users Group Information	6
Observations of the December 1987 Management Committee Meeting	7
From the <i>login</i> : Newsletter - Volume 12 Number 6	9
Dallas USENIX Conference	10
Scheduled Dallas Tutorials	10
Call for Papers: Summer 1988 USENIX Conference	11
Fifth Workshop on Real-Time Software and Operating Systems	12
cpio	13
Book Reviews	16
New System V Books	16
Portable C and UNIX System Programming	17
UNIX System Administration	20
<i>Computing Systems</i> - New USENIX Quarterly	21
2.10BSD Software Release Available	22
Future Meetings	23
Publications Available	23
French UNIX User's Group Conference	24
EUUG Spring 1988 Conference	24
From the EUUG Newsletter - Volume 7 Number 3	25
MINIX: A UNIX clone with source code	26
A User Programmable Telephone Switch	35
A Day in the Life of Owles Hall	49
Apologia	52
The X/OPEN Native Language System - Inside The Message Presentation	53
UNIX Standardisation: A Bystander's View	59

From the EUUG Newsletter - Volume 7 Number 3 <i>continued</i>	63
Demand Controlled Debug Logging	63
Book Review - UNIX System Programming	66
UNIX Throws Up	67
The EUUG Ditty	74
Call for Papers - 4th International Software Process Workshop	76
Report from ICEUUG	77
The Belgian UNIX systems Users Group	79
Colloquium on Interational Computer Networks in Belgium	81
News from the Netherlands	83
Planning Dates for the AFUU	85
News from UKUUG	86
EUUG Executive Committee Report	90
C++ Gossip	92
Status Report on the Draft Proposed ANSI/ISO C Standard	94
Book Review - The UNIX System V Environment	97
POSIX Progress at ISO Level and BSI Level	98
EUUG National Groups	101
UNIX Clinic	102
What's New With System V	104
EUnet	106
The X/OPEN Mid-Term Report	110
Book Review - troff typesetting for UNIX systems	112
Book Review - Document Formatting and Typesetting on the UNIX system	114
Letters to the Editor	116
AUUG Membership Catorgories	117
AUUG Forms	119

Copyright © 1987. AUUGN is the journal of the Australian UNIX* systems User Group. Copying without fee is permitted provided that copies are not made or distributed for commercial advantage and credit to the source must be given. Abstracting with credit is permitted. No other reproduction is permitted without prior permission of the Australian UNIX systems User Group.

* UNIX is a registered trademark of AT&T in the USA and other countries.

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

General Correspondence

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

The AUUG Secretary,
Department of Computer Science,
Melbourne University,
Parkville, Victoria 3052.
AUSTRALIA

ACSnet: auug@munnari.oz

AUUG Executive

Ken McDonell, *President*

kenj@moncsbruce.oz
Department of Computer Science, Monash University, Victoria

Robert Elz, *Secretary*

kre@munnari.oz
Department of Computer Science, University of Melbourne, Victoria

Chris Maltby, *Treasurer*

chris@gris.oz
Softway Pty. Ltd., N.S.W.

Chris Campbell, *Committee Member*

chris@olisyd.oz
Olivetti Australia, N.S.W.

Piers Lauder, *Committee Member*

piers@basser.cs.su.oz
Basser Department of Computer Science, Sydney University, N.S.W.

John Lions, *Committee Member*

johnl@elecvox.oz
School of Electrical Engineering and Computer Science, University of New South Wales, N.S.W.

Tim Roper, *Committee Member*

timr@labtam.oz
Labtam Limited, Victoria

Next AUUG Meeting

The next meeting will be held in Melbourne during September 1988.
Further details are provided in this issue.

AUUG Newsletter

Editorial

I hope you enjoy this issue and please contribute to the next issue.

REMEMBER, if the mailing label that comes with this issue is highlighted, it is time to renew your AUUG membership.

AUUGN Correspondence

All correspondence regarding the AUUGN should be addressed to:-

John Carey
AUUGN Editor
Computer Centre
Monash University
Clayton, Victoria 3168
AUSTRALIA

ACSnet: auugn@monu1.oz

Phone: +61 3 565 4754

Contributions

The Newsletter is published approximately every two months. The deadline for contributions for the next issue is Friday the 12th of February 1988.

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

I prefer documents sent to me by via electronic mail and formatted using *troff* -mm and my footer macros, troff using any of the standard macro and preprocessor packages (-ms, -me, -mm, pic, tbl, eqn) as well TeX, and LaTeX will be accepted.

Hardcopy submissions should be on A4 with 35 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. The current rate is AUD\$ 200 dollars per page.

Mailing Lists

For the purchase of the AUUGN mailing list, please contact Chris Maltby.

Disclaimer

Opinions expressed by authors and reviewers are not necessarily those of the Australian UNIX systems User Group, its Newsletter or its editorial committee.

President's Report

Whilst it may appear that the rumours of my death were exaggerated in the last AUUGN, a more rational and plausible explanation is that John Carey's production schedule has resulted in another issue going to press before my resignation takes effect on December 31.

Following the recent AUUG Management Committee Meeting, the arrangements for technical meetings are now settled. As of 1988, AUUG will move to a format featuring,

- One 3-day meeting and exhibition each year (probably in September) to be held in a major capital city (this is deliberately vague!). The initial schedule is,

Date	City	Place
1988, September 13-15	Melbourne	Southern Cross Hotel
1989	Sydney	
1990	Melbourne	

- Local AUUG Chapters are to be encouraged to hold smaller, more informal meetings each year in the February time-frame. To this end, AUUG is offering financial support to the Chapters to underwrite the costs associated with bringing invited speakers from interstate.

It is hoped that these arrangements will offer the undoubted benefits of a professionally run annual national meeting and exhibition, combined with the informality of small meetings, reminiscent of Unix user meetings from an earlier decade.

Details of other decisions taken at the Management Committee Meeting appear elsewhere in this issue.

Thanks to John Lions for resuming the Presidency from January 1 – I know he'll be well supported by all AUUG members.

Best wishes to all members in the coming months.

Finally, I'd like to thank all those whose efforts have made my term as AUUG President such an enjoyable one. May the true wisdom of line 2338 be revealed to each and every one of you; failing that, I hope you enjoy good health and avoid the dreaded "Cerebellum fault (code dumped)".

Ken J. McDonell

ps. my new e-mail address is kenj@pyramid.com

Adelaide UNIX Users Group

The Adelaide UNIX Users Group has been meeting on a formal basis for 12 months. Meetings are held on the third Wednesday of each month. To date, all meetings have been held at the University of Adelaide. However, it was recently decided to change the meeting time from noon to 6pm. This has necessitated a change of venue, and, as from April, meetings will be held at the offices of Olivetti Australia.

In addition to disseminating information about new products and network status, time is allocated at each meeting for the raising of specific UNIX related problems and for a brief (15-20 minute) presentation on an area of interest. Listed below is a sampling of recent talks.

D. Jarvis	"The UNIX Literature"
K. Maciunas	"Security"
R. Lamacraft	"UNIX on Micros"
W. Hosking	"Office Automation"
P. Cheney	"Commercial Applications of UNIX"
J. Jarvis	"troff/ditroff"

The mailing list currently numbers 34, with a healthy representation (40%) from commercial enterprises. For further information, contact Dennis Jarvis (dhj@aegir.dmt.oz) on (08) 268 0156.

Dennis Jarvis,
Secretary, AdUUG.

Dennis Jarvis, CSIRO, PO Box 4, Woodville, S.A. 5011, Australia.

PHONE: +61 8 268 0156 UUCP: {decvax,pesnta,vax135}!mulga!aegir.dmt.oz!dhj
 ARPA: dhj@aegir.dmt.oz!dhj@seismo.arpa
 CSNET: dhj@aegir.dmt.oz

Observations of the December 1987 Committee Meeting

Introduction

I have been asked by the AUUG Management Committee to attend their meetings and pass on to the members my observations in a timely manner.

A more formal record of the proceedings has been prepared by the Secretary and will appear in a following issue after ratification by the Committee.

So here are the major items discussed at the meeting held at University of Melbourne on Wednesday the 10th of December 1987.

New President for 1988

Ken McDonell has resigned as President as he will be working in the United States next year. This is effective as of December 31st 1987. The Committee elected John Lions to fill the vacancy until the AUUG elections are held next year.

Financial Position

The AUUG is in a healthy financial position with over 30K in the bank.

Secretarial Assistance

To improve service to members and to relieve some of the workload on the AUUG Executive it has been decided to investigate hiring secretarial assistance. The assistance would include answering enquiries about AUUG and doing the day to day administrative work.

Summer 87/88 Meeting Abandoned

The AUUG Meeting normally held early in the year has been abandoned for 1988.

This is due to the difficulties in finding the combination of a suitable venue and a willing host in time.

New Meeting Format for Winter 88 Conference

The next Conference of the AUUG will be held over three days in Melbourne during early September 1988. The meeting will follow the trend started by the successful Sydney Meeting which was to use a professional approach which included the use of sponsors and a professionally organized equipment exhibition. The Melbourne Conference will be organized by a professional conference organiser which will handle pre-conference publicity, equipment exhibit, registrations, and printing of programmes.

The suggested theme of conference is "Networking" and the Committee is approaching suitable overseas speakers for the occasion.

The format of future meetings will be a large professional technical meeting in Sydney or Melbourne in Winter each year and smaller informal meetings in the Summer.

Regional Meetings for Summer 88/89

Instead of the usual single meeting in early 1989, people will be encouraged by AUUG to hold one day technical meetings in their area. To help, AUUG may sponsor interstate speakers of note.

Incorporation

The application for incorporation of AUUG was resubmitted to Corporate Affairs in Victoria.

They are willing to incorporate after the Constitution was ammended by a members vote earier this year except for the use of the trademark UNIX in the name Australian UNIX systems Users Group Incorporated without written permission. In Victoria the trademark is held by AT&T Technologies.

It was decided to change the name to AUUG Incorporated or ask AT&T for permission to use the name.

AUUG and AUUGN will be registered as trademarks of the Group.

Constitutional Changes

Committee member, Tim Roper is organising proposed changes to the Constitution. These include a new position of Vice President, removal of Committee members, and possible changing the name of the Group to enable incorporation. These changes will be put to the members vote in the new year.

Members Benefits

It was decided the USENIX "Computer Systems Journal" will be provided to Institutional members as part of their subscription and available to members at cost of \$30 per annum.

At some time in the future a 4.3BSD manual distribution will be available through the AUUG.

ACSnet SIG

ACSnet Special Interest Group is looking at several ideas at improving the service offered by ACSnet. This includes establishing a 2400 bps leased line backbone between every major capital city; and forming a non-profit company, perhaps sponsored by AUUG, to run ACSnet. This would be similar to the USENIX UUNET company.

A proposal will be presented at February Committee Meeting.

AUUG Logo

A competition will be held for the design of a AUUG logo. The prize will be a year's free membership.

;login:

The USENIX Association Newsletter

Volume 12, Number 6

November/December 1987

CONTENTS

Dallas USENIX Conference	3
Scheduled Dallas Tutorials	3
Call for Papers: Summer 1988 USENIX Conference	4
Fifth Workshop on Real-Time Software and Operating Systems	5
cpio	6
Book Reviews	9
New System V Books	9
<i>Kevin W. Baranski-Walker</i>	
Portable C and UNIX System Programming	10
<i>John S. Quarterman</i>	
UNIX System Administration	13
<i>Rob Kolstad</i>	
Summary of the Board of Directors' Meeting	14
New Membership Rates Set	15
<i>Computing Systems</i> - New USENIX Quarterly	16
1988 Elections for Board of Directors	16
2.10BSD Software Release Available	17
Have You M-o-v-e-d?	17
4.3BSD Manuals Available to All Members	18
4.3BSD Manual Reproduction Authorization and Order Form	19
Future Meetings	20
Publications Available	20
French UNIX User's Group Conference	21
EUUG Spring 1988 Conference	21
Local User Groups	22

The closing date for submissions for the next issue of ;login: is January 3, 1988

USENIX THE PROFESSIONAL AND TECHNICAL
UNIX® ASSOCIATION

;login:

Dallas USENIX Conference

Rob Kolstad, Chair

Tuesday through Friday, February 9-12, 1988, mark the Winter 1988 USENIX Technical Conference in Dallas, Texas. The Grand Kempinski Hotel (formerly The Registry Hotel) will host the conference's two days of tutorials and two days of technical sessions.

There will be 21 tutorials, including several new ones. The two days of technical presentations will include two special half-day sessions on large-scale networks of workstations (as implemented in the Andrew project at Carnegie-Mellon University and MIT's Project Athena) and sessions devoted to new systems management techniques, the popular work-in-progress sessions late in the day, and other talks on state-of-the-art technical advances in UNIX and its applications.

The program committee is currently reviewing over 80 abstracts that were submitted for the conference. I am confident that the program will be an exceptionally strong and interesting one.

Conference information will be sent to all current members or may be obtained from:

Judy DesHarnais
USENIX Conference Office
P.O. Box 385
Sunset Beach, CA 90742
(213) 592-1381 or 592-3243
{uunet,ucbvax}!usenix!judy

See you in Dallas!

Scheduled Dallas Tutorials

TUESDAY

Introduction to 4.3BSD Internals
Introduction to UNIX System V Internals
UNIX System V Streams Device Driver
Managing A Local Area Network
Managing a Network of NFS Systems
Software Development Using C and UNIX
UNIX System V Remote File Sharing
X Window System Programming
An Introduction to C++
Sendmail, news, and uucp

WEDNESDAY

Advanced Topics on 4.3BSD Internals
Advanced UNIX System V Internals
UNIX Device Driver Design (4.2BSD)
4.xBSD System Administration
Language Construction Tools on the UNIX System
Special Topics in C
Open Network Computing and NFS
NeWS
An Introduction to 3D Computer Graphics
POSIX Implementation
The MACH Operating System

;login:

Call for Papers
Summer 1988 USENIX Conference
San Francisco
June 20-24, 1988

Papers in all areas of UNIX-related research and development are solicited for formal review for the technical program of the 1988 Summer USENIX Conference. Accepted papers will be presented during the three days of technical sessions at the conference and published in the conference proceedings. The technical program is considered the leading forum for the presentation of new developments in work related to or based on the UNIX operating system.

Appropriate topics for technical presentations include, but are not limited to:

- Kernel enhancements
- UNIX on new hardware
- User interfaces
- UNIX system management
- The internationalization of UNIX
- Performance analysis and tuning
- Standardization efforts
- UNIX in new application environments
- Security
- Software management

All submissions should contain new and interesting work. Unlike previous technical programs for USENIX conferences, the San Francisco conference is requiring the submission of **full papers** rather than extended abstracts. Further, a tight review and production cycle will not allow time for rewrite and re-review. (Time is, however, scheduled for authors of accepted papers to perform minor revisions.) Acceptance or rejection of a paper will be based *solely* on the work as submitted.

To be considered for the conference, a paper should include an abstract of 100 to 300 words, a discussion of how the reported results relate to other work, illustrative figures, and citations to relevant literature. The paper should present sufficient detail of the work plus appropriate background or references to enable the reviewers to perform a fair comparison with other work submitted for the conference. Full papers should be 8-12 single spaced typeset pages, which corresponds to roughly 20 double spaced, unformatted, typed pages. Format requirements will be described separately from this call. All final papers must be submitted in a format suitable for camera-ready copy. For authors who do not have access to a suitable output device, facilities will be provided.

Four copies of each submitted paper should be received by **February 19, 1988**; this is an absolute deadline. Papers not received by this date will not be reviewed. Papers which clearly do not meet USENIX's standards for applicability, originality, completeness, or page length may be rejected without review. Acceptance notification will be by **April 4, 1988**, and final camera-ready papers will be due by **April 25, 1988**.

Send technical program submissions to:

Sam Leffler (415) 499-3600
SF-USENIX Technical Program ucgvax!sfusenix
PIXAR
P.O. Box 13719
San Rafael, CA 94913-3719

FULL PAPERS ARE DUE FEBRUARY 19, 1988

;login:

**Fifth Workshop
On
Real-Time Software and Operating Systems**

**May 12-13, 1988
Omni-Shoreham Hotel
Washington, DC**

Sponsored by
The IEEE Computer Society
The USENIX Association

This year's workshop broadens the scope to include general real-time systems. This workshop will bring together researchers, designers, and implementers of real-time operating systems and software. There will be a substantial emphasis on practical experience, so workers from industrial organizations are encouraged to attend. Topics of specific interest include:

- Primary requirements of real-time systems
- Distributed real-time operating systems
- Application-specific operating systems
- Practical experiences and implications
- Exotic applications: medicine, music, etc.
- Architectural support for real-time
- Language, programming support, and reusability
- Types of real-time constraints
- Scheduling and resource management
- Predictability, adaptability, and maintainability
- Reliability and fault tolerance
- Instrumentation and performance measurement
- Case studies

The format of the workshop will be geared to encourage intense technical interactions and focussed discussions.

Attendance will be limited to between 75 and 100 active workers in the field. To participate in the workshop, please submit four copies of an extended abstract or position paper of up to 5 pages describing your current efforts to Lui Sha by **February 15, 1988**. The abstract should focus on insights and lessons gained from recent research and practical experience. Complete details regarding the workshop will be sent to all participants along with the acceptance letters by March 15, 1988. A digest of accepted abstracts will be made available to participants at the workshop.

General Chair

Professor John Stankovic
Department of Computer
and Information Science
Graduate Research Center
University of Massachusetts
Amherst, MA 01003
(413) 545-0720
stankovic@cs.umass.edu

Program Co-chair

Dr. Marc Donner
IBM Research
P.O. Box 218
Yorktown Heights, NY 10598
(914) 945-2032
donner@ibm.com

Program Co-chair

Dr. Lui Sha
Computer Science Department
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213
(412) 268-7668
sha@k.gp.cs.cmu.edu

;login:

cpio

In the the July/August 1987 issue of ;login: (vol. 12, No. 4), there was a lengthy discussion of tar vs. cpio.

This is the latest cpio proposal, as it appears in 1003.1 Draft 11 (except I've probably got the section numbers wrong). The "previous section" is, of course, USTAR, unchanged for some time.

John S. Quarterman

FOR COMPUTER ENVIRONMENTS

Std 1003.1-Draft 11

Editor's Note: The following section has been proposed as a replacement for, or an addition to, the previous section. The small group that considered the issue at the June 1987 meeting determined that indications were needed on how to extend this subsection to account for at least the following items:

- symbolic links
- contiguous files
- file name length
- i-node number size

There is a possibility that these concerns may be addressed by text in the Rationale, rather than in the body of the standard.

10.3.1 cpio Archive Format

The byte-oriented cpio archive format is a series of entries, each comprised of a header that describes the file, the name of the file, and then the contents of the file.

An archive may be recorded as a series of fixed size blocks of bytes. This blocking shall be used only to make physical I/O more efficient. The last group of blocks is always at the full size.

For the byte-oriented cpio archive format, the individual entry information must be in the order indicated and is described by:

Byte-Oriented cpio Archive Entry

Field Name	Header	
	Length	Interpreted as
c_magic	6 bytes	octal number
c_dev	6 bytes	octal number
c_ino	6 bytes	octal number

c_mode	6 bytes	octal number
c_uid	6 bytes	octal number
c_gid	6 bytes	octal number
c_nlink	6 bytes	octal number
c_rdev	6 bytes	octal number
c_mtime	11 bytes	octal number
c_namesize	6 bytes	octal number
c_filesize	11 bytes	octal number

File Name		
Field Name	Length	Interpreted as
c_name	c_namesize	pathname string

File Data		
Field Name	Length	Interpreted as
c_filedata	c_filesize	data

10.3.1.1 Header

For each file in the archive, a header as defined above shall be written. The information in the header fields shall be written as streams of bytes interpreted as octal numbers and shall be right-justified and zero filled. The fields shall be interpreted as follows:

• c_magic shall identify the archive as being a transportable archive by containing the magic bytes as defined by MAGIC ("070707").

• c_dev and c_ino shall contain values which uniquely identify the file within the archive (i.e., no files shall contain the same pair of c_dev and c_ino values unless they are links to the same file). The values shall be determined in an implementation defined manner.

• c_mode shall contain the file type and access permissions as defined in the tables below.

• c_uid shall contain the user id of the owner.

• c_gid shall contain the group id of the group.

• c_nlink shall contain the number of links referencing the file at the time the archive was created.

• c_rdev shall contain implementation defined information for character or block special files.

;login:

• `c_mtime` shall contain the latest time of modification of the file.

• `c_namesize` shall contain the length of the path name, including the terminating null byte.

• `c_filesize` shall contain the length of the file. This is the length of the data section following the header structure.

10.3.1.2 File Name

`c_name` shall contain the path name of the file. The length of the name is determined by `c_namesize`; the maximum length of this string is 256 bytes.

10.3.1.3 File Data

Following `c_name`, there shall be `c_filesize` bytes of data. Interpretation of such data shall occur in a manner dependent on the file. If `c_filesize` is zero, no data shall be contained in `c_filedata`.

10.3.1.4 Special Entries

Special files, directories, and the trailer are recorded with `c_filesize` equal to zero. The header for the next file entry in the archive shall be written directly after the last byte of the file entry preceding it. A header denoting the file name "TRAILER!!!" shall indicate the end of the archive; the contents of bytes in the last block of the archive following such a header are undefined.

10.3.1.5 cpio Values

Values needed by the `cpio` archive format are described as follows:

Values for `c_mode` field
File permissions

Name	Value	Indicates
<code>C_IRUSR</code>	000400	read by owner
<code>C_IWUSR</code>	000200	write by owner
<code>C_IXUSR</code>	000100	execute by owner
<code>C_IRGRP</code>	000040	read by group
<code>C_IWGRP</code>	000020	write by group
<code>C_IXGRP</code>	000010	execute by group
<code>C_IROTH</code>	000004	read by others
<code>C_IWOTH</code>	000002	write by others
<code>C_IXOTH</code>	000001	execute by others
<code>C_ISUID</code>	004000	set uid
<code>C_ISGID</code>	002000	set gid
<code>C_ISVTX</code>	001000	reserved

Values for `c_mode` field

Name	File type	
	Value	Indicates
<code>C_ISDIR</code>	040000	directory
<code>C_ISFIFO</code>	010000	FIFO
<code>C_ISREG</code>	100000	regular file
<code>C_ISBLK</code>	060000	block special
<code>C_ISCHR</code>	020000	character special
	110000	reserved
	120000	reserved
	140000	reserved

`C_ISDIR`, `C_ISFIFO`, and `C_ISREG` shall be supported on a POSIX conforming system; additional values defined above are reserved for compatibility with existing systems. Additional file types may be supported; however, such files should not be written on archives intended for transport to portable systems.

10.3.1.6 References

<grp.h> §9.2.1, <pwd.h> §9.2.2, <sys/stat.h> §5.6.1, `chmod()` §5.6.4, `link()` §5.3.4, `mkdir()` §5.4.1, `read()` §6.4.1, `stat()` §5.6.2.

The following represents the position of X/Open on cpio. - PHS

From: Jim R Oldroyd
<mcvax!inset!jr@seismo.css.gov>
Date: Wed Jul 15 18:26:19 BST 1987
Organization: The Instruction Set Ltd.
To: jsq@longway.tic.com
Subject: Benefits of cpio over tar

I would like to present a number of points regarding `cpio` which I feel are relevant to the ongoing discussion concerning a Data Interchange Format for the POSIX 1003.1 standard.

I shall correct a number of important points regarding the `cpio` format; points which have been incorrectly stated in recent articles.

1. At no time has a proposal been made to standardise the binary `cpio` format. Only the `cpio -c` format is under consideration.

2. The `cpio -c` format is widely in use in Europe for both Data Interchange and Archival purposes. Its widespread use can be attributed, in part, to its endorsement by the X/OPEN Group.

;login:

3. Only one version of the `cpio -c` format is currently in use. It is this format being proposed for standardisation.

4. The `cpio -c` header is written entirely in character form. No numerical information is stored in machine-dependent binary form.

5. The `cpio -c` format is capable of archiving and restoring all POSIX file types: directories, block special files, character special files, regular files and fifos.

6. The `cpio -c` format can handle pathnames up to 256 bytes. This is the length guaranteed on all POSIX systems.

7. The `cpio -c` format is in the public domain. (See X/OPEN Portability Guide, Volume 2, `cpio(4)`).

8. Inode numbers are not recorded. Symbolic values (derived from a file's inode and device numbers) are stored in the header block. These values are used solely for hard link resolution.

9. File types are stored in symbolic form. Symbols are derived from historical UNIX file type values. There is room for 64 file types; currently only 5 are supported.

A number of points have recently been raised as drawbacks of `cpio`. These points seem to be problems with a particular implementation of a `cpio` utility. As the characteristics of the utility are not relevant for 1003.1, I present only a short summary of points:

file names are terminated by `'\0'`:

This is normal UNIX practice for string termination and applies to `tar` (and USTAR) equally. On `cpio`, the `'\0'` is redundant information and need not be interpreted as the file name length is also provided.

the user interface is less convenient:

This is subjective; many people feel that the opposite is true. The user interface is easily alterable (discuss with 1003.2).

file name size is 128 bytes:

Wrong! It is 256; see above.

`cpio` header is full of OS dependent information:

Wrong! All information describes file characteristics. There is no OS dependent information. See point 9, above.

header must start on a word boundary:

Wrong! The header is character oriented and can be read as individual bytes from the archive.

format cannot be extended to meet future requirements:

Wrong! Implementations already exist which can archive symbolic links and contiguous files. There is far more scope for future extension than available in the proposed USTAR format.

Independent of the archive format used, some guidelines must be followed to ensure that an archive can be extracted on ANY POSIX system. Note that the following are NOT rules for using `cpio`; they apply equally well to other interchange formats if portability across ALL systems is to be achieved:

- only POSIX defined file types should be archived
- headers should be written in US ASCII character set
- minimum values in section 2.9 for `h_uid`, `h_gid`, `h_nlinks`, etc. should not be exceeded
- no portion of any filename should exceed 14 characters
- one `cpio` archive should fit on a single medium
- only one archive should exist per medium
- relative pathnames (ie, no leading `/`) should be used
- tapes should be written in "raw" mode
- tapes should be written with 5120 byte blocks

Any archive intended for use only between systems supporting more capabilities than the minimum required by POSIX need not be so restrictive.

I believe that the `cpio -c` tape format has a number of strong advantages over both the existing `tar` and the POSIX extended `tar` formats. The `cpio -c` format handles all POSIX file types correctly, it has been extended to handle other known file types and there is adequate opportunity for further extension.

Thank you,
Jim R Oldroyd

Book Reviews

New System V Books

Reviewed by Kevin D. Baranski-Walker

University of California - Berkeley
kevin@violet.Berkeley.EDU

This review will take a look at two recent series of works on the AT&T System V Release 3 of UNIX. More precisely this is a review of a re-packaging of the familiar AT&T reference manuals and a look into the latest System V related readings from The Waite Group. The books reviewed are:

American Telephone and Telegraph; Prentice-Hall publishers:

- UNIX System V Utilities Release Notes, AT&T
- UNIX System V Programmer's Reference Manual, AT&T
- UNIX System V Network Programmer's Reference Guide, AT&T
- UNIX System V Streams Programmer's Guide, AT&T
- UNIX System V Streams Primer, AT&T
- UNIX System V User's Reference, AT&T
- UNIX System V User's Guide, 2nd Ed., AT&T
- UNIX System V Programmer's Guide, AT&T

The Waite Group; Howard W. Sams & Company publishers:

- UNIX System V Primer (Revised Edition), Waite, Prata and Martin
- UNIX System V Bible (Commands and Utilities), Prata and Martin

Prentice-Hall has acquired the publication rights for the AT&T System V Release 3.0 series of guides and reference manuals. This marks a departure from what has become a tradition in the publication of computer system related documents. Typically the manufacturer of the software produces their documentation within their own facilities. Beginning with this series AT&T has granted exclusive publication rights to Prentice-Hall. Prentice-Hall is scarcely novice in the publication of computer science or computer industry related texts. Their most notable (at least in

sheer volume of sales) is the *C Programming Language*, by Kernighan and Ritchie. Clearly the advantage for a software developer to enlist the services of a conventional book publisher to package and market their documentation, is to broaden the distribution channels and to provide ready access to potential readers that may otherwise be difficult. If indeed this is the rationale for AT&T and Prentice-Hall to enter into such an arrangement, then the goal may well have been reached.

Aside from the production changes of this series, which are readily evident, very little in terms of content has changed from the previous printing. The contents of the *User's Reference Manual*, *Programmer's Reference Manual* and *Utilities Release Notes* in particular remain entirely unchanged from the original AT&T printing. A notable omission in these three manuals is the inclusion of useful indices and glossary. Inasmuch as these handbooks are directed at novice users, as well as referral materials for experts, such an oversight exacerbates many of the long held grievances of the UNIX documentation.

The *Streams Primer* and *Streams Programmer's Guide* reflect a refreshing change from the previous printings. Diagrams and examples are abundant and well suited. Once again, though, the indexing is lackluster in the case of the programmer's guide and again non-existent in the primer.

Though the *Network Programmer's Guide* maintains a similar flavor to the Streams primer and guide, it provides a sensible balance between the needs of the neophyte reader and the more experienced network programmer. Appendix C offers several beneficial examples that detail varying network models with exceptional clarity. This volume is mandatory for any System V user.

Very few assumptions are made about the reader in the *User's Guide*. It presents a fairly focused overview of the new user's approach to UNIX and System V in particular, yet can still function as a useful reference guide for the experienced. As with many introductory texts on computer related subjects, the *User's Guide* indulges in a liberal discussion on the

;login:

characterization of the user's relation to the system. Once the user has a grasp on these basics, the essentials of file accessing and manipulation (both command line and editing), shell programming, mail, and networking overviews are presented. Each section ends with several exercises, some merely to reinforce your just concluded reading while others provide useful trials in solving increasingly complex tasks. The User's Guide concludes with a quick reference command summary for the most used System V commands, `ed`, `vi` and shell programming commands. As with the Network guide a glossary is provided.

The *Programmer's Guide* is directed at application programmers and as with the Network Programmer's Guide delivers a very careful balance between the needs of the occasional, novice user and that of the experienced programmer. The seventeen chapters are rather comprehensive with special attention paid to the System V support tools; `awk`, `lex`, `yacc`, `curses`, `make`, `SCCS`, `sdb` and `lint`. Additionally, excellent tutorials on file and record locking, shared libraries and interprocess communication accompany the system tools section.

Howard W. Sams & Company are best known for publishing hobbyist oriented handbooks and reference materials (including numerous "How-To" and repair guides). Lately Sams has broadened their repertoire to include several MS-DOS, XENIX, C language and UNIX titles. Additionally they have merged the UNIX System Library books from Hayden Book publishers. A good portion of these combined titles have been authored or co-authored by The Waite Group. In the *UNIX System V Primer*, Waite, Martin and Prata have produced a very comprehensive indoctrination for the new System V user, similar to their previous introductory work on UNIX, "UNIX Primer Plus." Unfortunately this primer is so awash with innane comic illustrations and trite examples that it's difficult to recommend. The authors direct this book at "... a secretary or a manager in an office, or student in a computer science class, or a computer hobbyist who is interested in UNIX ...". For such an all-encompassing audience a slightly less jocular forum would have been in order. Presentation aside, this book does serve

well as a primer on System V UNIX (indeed the introduction is relevant to all UNIX novices). The diagrams, charts, sidebars and later examples are concise and well thought. Of particular note is chapter 9, "The UNIX Shell: Command Lines, Redirection, and Shell Scripts." This chapter offers a fine tutorial on the user's interaction with and control of the system.

The *UNIX System V Bible (Commands and Utilities)* is a quality companion to the AT&T User's Reference and User's Guide. This text has numerous useful examples and parallels the structure of the standard reference manual. A special section has been added entitled "UNIX Features" which is an excellent summary of the System V nuances.

Portable C and UNIX System Programming

by J. E. Lapin [Rabbit Software]
(Prentice-Hall)

Review by John S. Quarterman

Texas Internet Consulting
uunet!longway!jsq

This is a useful book. It fits in the gap between the existing literature about C and the little there is about writing portable UNIX programs[1,2]. The authors (of which there are actually eight) give a good impression right away by noting in the Preface that "no such thing as an 'unimportant' change was assumed" in their comparisons of functional differences in the programming interface. They have clearly put years of effort into this book, and they even supply a uucp mail address for comments. There are still problems, some of them serious, but first the best points.

Chapter 2, "Portable C Standard," about Rabbit Software's internal guidelines (not to be confused with X3J11's C Language Standard), is enlightened. Not only are the guidelines sensible (such as their comments on the comma operator in 2.12.2 and on the

;login:

goto statement in 2.13.1), organized (into Portability Rules, Maintenance Rules, Stylistic Guidelines, and Performance Techniques), and mostly complete, but the principles for choosing the rules are themselves admirable. Their method for handling multiple modules (2.14, also 2.3) seems a bit peculiar at first, but will be obvious to readers familiar with Modula-2.

There are a couple of problems with this material. It is good that the authors recognize the usefulness of a uniform textual style for programs, but they don't seem to be aware of the existing de facto standard, the "Indian Hill Style Sheet," which rather closely matches the basic style of most software written by the Bell Labs Research group or UC Berkeley Computer Science Research Group. Personally, I find Lapin's choice of positioning of braces to be the ugliest of all possibilities.

A more serious problem (perhaps unavoidable due to the publishing schedule) with this material and the book in general is that the last revisions referenced of the X3.159 C Language Standard by the ANSI X3J11 committee or the IEEE 1003.1 Standard for Portable Operating System Interface for Computer Environments (POSIX) were those of November 1985. Many things have changed since then. X3J11 has supplied a Rationale appendix to their standard which, among other things, clarifies historically obscure points regarding scope and storage class; that information could have helped in 2.3. Nonetheless, the book is generally accurate in its information regarding the C language.

Chapter 5, "Maintaining Portable Systems," has a good bit of practical advice clearly derived from experience, and could save programmers many mistakes. There is a very good list in 5.2.3 of recommended make productions that every makefile should support. Useful but often overlooked rules are collected and organized, for example, that only the Bourne shell should be used for portable shell scripts (5.4.1). Unfortunately, while the authors explicitly recommend against the use of the C shell for such scripts, they don't warn about the Korn shell. Use of the last is actually more insidious, since scripts written for it look so much like Bourne shell scripts, but use extensions that won't work with the Bourne shell. The authors mention SCCS (5.2), but fail to mention RCS.

The book compares a plethora of UNIX variants, including Version 7, System III, System V, 4.1BSD, 4.2BSD, 4.3BSD, XENIX 2.3, 3.0, and 5.0. It is admirable that this has been attempted (especially that the XENIX versions were included); there is much useful information presented, and clearly most of the work on the book went into the UNIX comparisons. Unfortunately, the material on comparisons of UNIX dialects is not as good as that on the C language.

Some historical information on the UNIX System in 1.5 is inaccurate and misleading. This is as much because of placement of information as because of what is actually said. The Programmer's Work Bench (PWB) is not mentioned in its historical period (1.5.1), even though it is one of the principal ancestors of System III and System V.

Bill Joy is first mentioned as working at Sun Microsystems (1.5.4), with no clear evidence given that he ever worked at Berkeley (1.5.2), nor of the PDP-11 Berkeley releases that preceded the Berkeley VAX work, nor of the paging system that was the first major part of that work. There is no mention whatever of DARPA, the government agency that funded much of the BSD work and strongly influenced its research and networking orientation. The only place networking is mentioned is when Sun's Network File System (NFS) is misidentified (1.5.3) as "the Networked File Standard (NFS) by a consortium of influential 4.2BSD vendors."

One could easily get the impression from the book that Dennis Ritchie and Ken Thompson are the principal developers of System V[3]. Their Bell Laboratories research system, V8 or Eighth Edition, is misdiagrammed (3.1) as being derived directly from Version 7, when the kernel, at least, was derived from 4.1BSD. (Their current system is Ninth Edition.)

The authors appear to have only seen a draft ("UniForum Draft Standard" is mentioned in 1.5.3) of the /usr/group 1984 Standard, even though other information was incorporated into the book at least as late as the end of 1985, according to the bibliographic information given.

The comments made (1.5.4) about the essential equivalence of X3J11's C Language Standard, the System V Interface Definition

(SVID), and the IEEE P1003 work may have been correct in 1985, but are clearly not true today. X3J11's Rationale states that the C Standard is derived from the Kernighan and Ritchie book and from the /usr/group 1984 Standard. The IEEE 1003.1 Trial Use Standard incorporated major features from 4.2BSD, and more recent drafts include more such features. Details on relations among these standards, System V, and 4.3BSD, may be found in a recent publication by /usr/group[4].

There are also a number of facilities in 1003.1 that do not exactly match any existing system, which is not to say that Lapin's basic point that progress is being made on real, international C and UNIX programming interface standards is wrong, but the way it is actually happening is not quite as the book predicted. For example, X/OPEN, the originally European-only group of computer manufacturers (1.5.4), has expanded to include U.S. companies and has revised their earlier strict adherence to the SVID in favor of conformance to the eventual IEEE 1003.1 Full Use Standard.

System V Release 3 and Issue 2 of the SVID include several features, such as the `rename()`, `mkdir()`, and `rmdir()` system calls, that were derived from the IEEE 1003.1 Trial Use Standard, which in turn got them from 4.2BSD. The most noticeable such facility is the directory manipulation routines (`opendir()`, `readdir()`, `closedir()`). Although this means that those routines have become a de facto standard and will probably come to be found on most UNIX variants, Lapin's inclusion of code in Appendix D which emulates them on old-style file systems is still a public service.

The main problem with the material on UNIX variants (not only the historical information, but also the actual comparisons) is a shortage of information about BSD systems[5,6]. Although the authors state (3.4.2) that 4.1BSD was their home system while most of the book was being prepared, they lump 4.1BSD and 4.2BSD together, with no mention of many of the profound differences between them, such as the networking facilities, inter-process communication, `select(2)`, long file names, symbolic links, scatter/gather I/O, more accurate timers, etc. It is true that many of these facilities are not

appropriate for inclusion in a book about portability among UNIX variants, because other variants often don't have these features at all. Yet, many System V-based systems have had the networking (and other) facilities from 4.2BSD added, and many facilities from BSD systems have been adopted by System V and elsewhere. These include not only the ones mentioned in the previous paragraph, but also such things as `catman`, which the book misidentifies (3.5) as originating in System V Release 2. There are also errors of omission, as when the 4.2BSD `restore` program is discussed (3.6), but the text doesn't mention that the original reason for the rewrite from the Version 7 `restor` program was to allow restoring through the ordinary file system system calls rather than by writing directly to the raw disk device.

Nonetheless, most of the comparison information is accurate and amazingly exhaustive: there are not only tables of comparison for most commands, system calls, and library routines, there are even tables of comparison for options of commands where that is useful. Finally let's not forget the occasional drollery: documenters who omit games will reincarnate as "worker insects of some sort."

This book is useful as it is now. I recommend buying it, while waiting for the authors to update it.

1. Chambers, John B., and Quarterman, John S., "UNIX System V and 4.1C BSD," Proceedings of Summer 1983 Toronto USENIX Conference, pp. 267-291, USENIX Association, P.O. Box 2299, Berkeley, CA 94710, 14 July 1983.
2. Uniejewski, Joseph, "UNIX System V and BSD4.2 Compatibility Study," Apollo Computer Inc., Chelmsford, MA 01824, March 28, 1985.
3. Bach, Maurice J., *The Design of the UNIX Operating System*, Prentice-Hall, Englewood Cliffs, NJ, 1986.
4. /usr/group, "POSIX Explored," /usr/group, 4655 Old Ironsides Drive, Suite 200, Santa Clara, CA 95050, 408-986-8840, 30 pp., October 1987.

;login:

5. Quarterman, J. S., Silberschatz, A., and Peterson, J. L., "4.2BSD and 4.3BSD as Examples of the UNIX System," ACM Computing Surveys, Volume 17, Number 4, pp. 379-418, December 1985.

6. Leffler, S. J., McKusick, M. K., Karels, M., Quarterman, J. S., The Design and Implementation of the 4.3BSD UNIX Operating System, Addison-Wesley, Reading, MA, 1988.

UNIX System Administration

by Frank Burke

(ISBN 0-15-593025-7, LC #86-70500)

Reviewed by Rob Kolstad

Convex Computer Corp.
convex!kolstad

This book's primary objective is "to educate UNIX system administrators." It is intended to be used in the fourth semester of a computer science degree program in which students have some degree of familiarity with UNIX. It succeeds in this singular objective - the book is a fine one for a college environment.

It is, however, a UNIX SYSTEM V administrator's guide. Owners of 4.xBSD systems will find only minimum utility here. It is surprising that administration techniques can be so different between AT&T and Berkeley UNIX. As stated in the summary, it is also only an introduction, not a comprehensive reference guide.

Thirteen chapters take the student through a "UNIX Computer Center" (complete with "UNIX operators"), login administration, teletype administration, file system administration, process administration, operations administration, security comments, system tuning, configuration, system generation, uucp network administration, and update administration.

The book includes many examples, but of course, it's impossible to have too many examples in this kind of presentation. The procedures set forth in numbered steps can greatly aid the novice system administrator in accomplishing his or her various goals.

While mastering the book may allow a student to begin duties as a system manager (a big step up when compared to former schema common in the just-bought-a-UNIX-box community), the book by no means teaches the becoming of an expert system administrator. Why? I suspect the reason relates to the book's intended purpose of a single semester course. Only seven column-inches (four inch columns) describe `fsck`. While a short discussion summarizes the file system structure, it is not possible to learn really effective use of the `fsck` program from the description.

Backups receive a similarly cursory treatment. While the commands to create a backup tape are listed, the "big picture" of a backup system does not appear. Again, the limited scope of the book may have precluded more detailed treatment.

In general, the book presents at least one example of just about every System V system administration feature. The motivation for the example may be brief - sometimes *too* brief. Unfortunately, the book will give no aid to those administrators joining the world of local area networking (TCP/IP and ethernet or STARLAN, for example). It must be considered as a very general introductory textbook.

Schools in the System V fold may wish to use it for a textbook if augmented by a practicum/laboratory. The Instructor's Guide (accompanying the text) provides two pages of suggestions for such a laboratory, but they're brief, e.g.: "Students learn about a dial-up network by studying uucp commands and files."

Harcourt Brace Jovanovich publishes the book at about \$16. The publisher says it is best purchased at a local bookstore; college bookstores are best.

;login:

Computing Systems – New USENIX Quarterly

There have been a number of queries about the new USENIX journal

Computing Systems

which will be published beginning February 1988. Michael D. O'Dell, Maxim Technologies, will serve as Editor-in-Chief.

Computing Systems will be published by the University of California Press. It will appear under the aegis of the USENIX Association, with the cooperation of the EUUG. The journal will be a membership benefit for members of USENIX and available at a reduced rate of \$20/year to those EUUG, AUUG, (or other national UNIX-user group) members who wish to receive it. It will be available by subscription through the University of California Press at \$40/year.

Computing Systems will be dedicated to the analysis and understanding of the theory, art, design, engineering, and implementation of advanced computing systems, with an emphasis on systems inspired or influenced by

the UNIX tradition. Articles concerning operating systems, architecture, networking, programming languages, and sophisticated applications are of interest. Papers will reflect a mix of theory and practical experience.

Computing Systems may, from time to time, also publish non-research articles concerning computing controversies and review articles concerning important books or papers.

Submissions, in n/troff format should be sent to `{uunet,ucbvax}!usenix!journal`. Hard copy submissions should be supplied in five (5) copies and mailed to

Computing Systems
USENIX Association
P.O. Box 2299
Berkeley, CA 94710

The Association hopes to have a rapid turnaround, with only 6-8 months between submission and publication.

;login:

2.10BSD Software Release Available

The USENIX Association and the Computer Systems Research Group (CSRG) of the University of California, Berkeley, are pleased to announce the distribution of a new release of the "Second Berkeley Software Distribution" (2.10BSD).

This release will be handled by the USENIX association, and is available to all V7, System III, System V, and 2.9BSD licensees. The Association will continue to maintain the non-profit price of \$200, as was charged by the CSRG. The release will consist of two 2400 foot, 1600 bpi tapes (approximately 80Mb) and approximately 100 pages of documentation. If you require 800 bpi tapes, please contact USENIX for more information.

The tape that USENIX will be distributing for the first few weeks will **only** support machines with split I/D and floating point hardware. This is not because any work remains to be done, but because we just haven't had the time to build and test a system.

Sites wishing to run 2.10BSD should also be aware that the networking is only lightly tested, and that certain hardware has yet to be ported. Contact Keith Bostic at the address below for current information as to the status

of the networking. As of August 6, 1987, the complete 4.3BSD networking is in place and running, albeit with minor problems. The holdup is that only the Interlan Ethernet driver has been ported, as well as some major space constraints. Note, if we decide to go to a supervisor space networking, 2.10 networking will only run on:

11/44/53/70/73/83/84
11/45/50/55 with 18 bit addressing

If you have questions about the distribution of the release, please contact USENIX at:

2.10BSD
USENIX Association
P.O. Box 2299
Berkeley, CA 94710

+1 415 528-8649
{uunet,ucbvax}!usenix!office

If you have technical questions about the release, please contact Keith Bostic at:

{ucbvax,scismo}!keith
keith@okeeffe.berkeley.edu
+1 415 642-4948

Keith Bostic
Casey Leedom

;login:

Future Meetings

USENIX 1988 Winter Conference and UniForum - Dallas

The USENIX 1988 Winter Conference, featuring tutorials and technical sessions, will be held on February 9-12, 1988, at the Grand Kempinski Hotel in Dallas, Texas. It will be concurrent with UniForum 1988, which will also be in Dallas.

AFUU UNIX Convention 88 Paris, March 7-10, 1988

EUUG Spring Conference London, April 11-15, 1988

USENIX 1988 Summer Conference and Exhibition - San Francisco

The USENIX 1988 Summer Conference and Exhibition will be held on June 20-24, 1988, at the Hilton Hotel in San Francisco, California. There will be a conference, tutorials, and vendor exhibits.

Long-term USENIX Conference Schedule

Feb 8-12	'88	Grand Kempinski, Dallas
Jun 20-24	'88	Hilton Hotel, San Francisco
Jan 31-Feb 3	'89	Town & Country Inn, San Diego
Jun 12-16	'89	Hyatt Regency, Baltimore
Jan 23-26	'90	Washington, DC
Jun 11-15	'90	Marriott Hotel, Anaheim
Jan 22-25	'91	Dallas
Jun 10-14	'91	Opryland, Nashville

Publications Available

The following publications are available from the Association Office. Prices and overseas postage charges are per copy. California residents please add applicable sales tax. Payments **must** be enclosed with the order and **must** be in US dollars payable on a US bank.

The EUUG Newsletter, which is published four times a year, is available for \$4 per copy or \$16 for a full-year subscription.

The July 1983 edition of the EUUG Micros Catalog is available for \$8 per copy.

Conference and Workshop Proceedings

Meeting	Location	Date	Price	Overseas Mail	
				Air	Surface
Graphics Workshop IV	Cambridge	October '87	\$10	\$15	\$5
USENIX	Phoenix	Summer '87	\$20	\$25	\$5
USENIX	Wash. DC	Winter '87	\$20	\$25	\$5
Graphics Workshop III	Monterey	December '86	\$10	\$15	\$5
USENIX	Atlanta	Summer '86	\$10	\$25	\$5
USENIX	Dallas	Winter '85	\$10	\$25	\$5
Graphics Workshop I	Monterey	December '84	\$ 3	\$ 7	\$5

;login:

French UNIX User's Group Conference

Paris

March 7-10, 1988

The French Association of UNIX Users (AFUU) in cooperation with the Bureau International de Relations Publiques is organizing a conference in Paris, 7-10 March 1988.

There will be tutorials on the first day and technical meetings and an exhibition running concurrently the next three days.

While UNIX Convention 88 is intended to be a primarily French event, it is expected that a considerable number of overseas visitors will participate.

The chair of the Program Committee is Christophe Binot. Information is available from:

AFUU c/o SUPELEC
Attn.: Anne Garnery, Convention UNIX 88
Plateau de Moulon
91190 Gif-sur-Yvette
FRANCE

mcvax!inria!afuu!anne

EUUG Spring 1988 Conference

London

April 11-15, 1988

The UKUUG will host the Spring '88 European UNIX systems User Group Technical Conference at the Queen Elizabeth II Conference Center in London. Technical tutorials will be held on April 11 & 12, followed by the three day conference. A pre-conference registration pack will be issued in early December, 1987.

For further information, contact the EUUG Secretariat at:

EUUG
Owles Hall
Buntingford, Herts. SG9 9PL
United Kingdom

Phone: (+44) 763 73039
Fax: (+44) 763 73255 (G2)

EUROPEAN UNIX SYSTEMS USER GROUP NEWSLETTER

*Volume 7
Number 3*

Editorial	1
MINIX: A UNIX Clone with source code	3
Conference Announcements	12, 59, 63, 79
A User Programmable Telephone Switch	13
A Day in the Life of Owles Hall	27
Apologia	30
The X/OPEN Native Language System	31
UNIX Standardisation: A Bystander's View	37
Demand Controlled Debug Logging	41
Book Reviews	44, 76, 97,99
UNIX Throws Up	45
The EUUG Ditty	52
Call for Papers	54
Report from ICEUUG	55
The Belgian UNIX Systems User Group	57
News from The Netherlands	61
News from UKUUG	65
EUUG Executive Committee Report	69
C++ Gossip	71
Status Report on the Draft Proposed ANSI/ISO C Standard	73
POSIX Progress at ISO Level and BSI Level	77
EUUG Tape Distributions	81
UNIX Clinic	87
What's New with System V	89
EUnet	91
The X/OPEN Mid-term Report	95
Product Announcements	96, 98
Glossary	101

MINIX: A UNIX clone with source code

March 1987

Andrew S. Tanenbaum
ast@cs.vu.nl
...!mcvax!blotter!ast

Dept. of Mathematics and Computer Science
Vrije Universiteit
Amsterdam, The Netherlands

MINIX is a complete rewrite of UNIX. Neither the kernel nor the utility programs contains any AT&T code, so the source code is free of the AT&T licensing restrictions and may be studied by individuals or in a course. The system runs on the IBM PC, XT, or AT, and does not require a hard disk, thus making it possible for individuals to acquire a UNIX-like system for home use at a very low cost.

Internally, MINIX is structured completely differently from UNIX. It is a message passing system on top of which are memory and file servers. User processes can send messages to these servers to have system calls carried out. The paper describes the motivation and intended use of the system, what the distribution contains, and discusses the system architecture in some detail.

1. Introduction

When AT&T first licensed UNIX outside of Bell Laboratories, it was widely studied in operating systems courses at universities (and in industry). Prof. John Lions of the University of New South Wales in Australia even wrote a little booklet providing a commentary on the source code, which for the most part was comment-free. Lions' booklet plus the UNIX source code made it possible for students to get hands-on experience working with, and modifying the code of a real operating system.

With the advent of Version 7, AT&T decided to put an end to the teaching of UNIX to students, and added a clause to the standard university contract prohibiting use of the source code in the classroom. Since that time, professors and students have largely had to be content with operating systems theory because no system that was small enough to be understandable yet large enough to be realistic has been available in source form.

To remedy this situation, several years ago I decided to write a new operating system from scratch that would be system-call compatible with UNIX but completely new inside. In addition to eliminating the licensing problems, this system would be written using modern software concepts such as structured programming, modularity, and file servers. UNIX itself was begun in the early 1970s when the main design issue was squeezing it onto a PDP-11, rather than making the code easy for others to read.

That work is now complete and has resulted in a system called MINIX (mini UNIX) because it leaves out some of the more esoteric system calls in an attempt to make the system smaller and easier to understand. MINIX was originally written for the IBM PC, XT, and AT, but work is currently under way to port it to the 68000 and other computers. The system is written in C and some care has been taken in the design to make the port to small computers without memory management hardware as straightforward as possible. This will be discussed in detail later in the paper.

To avoid confusion, it is worthwhile stating explicitly who MINIX is aimed at. There are two potential groups of users.

1. Professors, students, and others who are interesting in legally obtaining and studying the source code of a UNIX-like operating system.
2. People who would like to run a UNIX-like system (especially at home), but have not been able to afford it. Since MINIX does not require a hard disk and the complete system, including both the binaries and the sources costs under \$80, the set of potential users is much larger than for UNIX.

Thus MINIX does not really compete with UNIX. Rather, it fills a niche that is currently unoccupied.

For the first category (i.e. educational) users, several options have been provided. The system can be modified and maintained on an IBM PC with or without a hard disk, using itself, a version of UNIX, or even MS-DOS. It can also be cross compiled on a VAX or other time-shared computer running UNIX. Furthermore, the software distribution contains an interpreter for the IBM PC (including I/O) so that the resulting system can be run on a VAX or other computer, preferably a fast one, in case no real IBM PCs are available for students. The MINIX file system can also be modified and run on almost any computer, since it is structured as a free-standing file server. The file server can also be used in a network of diskless workstations.

For the second category (i.e. impoverished) users, several versions of the system have been configured. The normal ones run on 640K PCs with two floppy disks or 512K ATs with one floppy disk, but a special version has also been configured for 256K PCs with only one floppy disk. This version does not contain the C compiler, but is otherwise complete.

MINIX is system-call compatible with Version 7 UNIX for both practical and ideological reasons. On the practical side, I was unable to figure out how to make either 4.3 BSD or System V run on a 256K IBM PC with only 1 floppy disk. On the ideological front, many people (myself included) strongly believe that Version 7 was not only an improvement on all of its predecessors, but also on all of its successors, certainly in terms of simplicity, coherence and elegance. Users who prefer features to elegance should program in Ada[†] on a large IBM mainframe running MVS.

MINIX implements all the V7 system calls, except ACCT, LOCK, MPX, NICE, PHYS, PKON, PKOFF, PROFIL, and PTRACE. The other system calls are all implemented in full, and are exactly compatible with V7. In particular, FORK and EXEC are fully implemented, so MINIX can be configured as a normal multiprogramming system, with several background jobs running at the same time (memory permitting), and even multiple users.

[†] Ada is a Registered Trademark of the U.S. Dept. of Defense

The MINIX shell is compatible with the V7 (Bourne) shell, so to the user at the terminal, running MINIX looks and feels like running UNIX. Over 70 utility programs are part of the software distribution, including `ar`, `basename`, `cat`, `cc`, `chmem`, `chmod`, `chown`, `cmp`, `comm`, `cp`, `date`, `dd`, `df`, `echo`, `grep`, `head`, `kill`, `ln`, `login`, `lpr`, `ls`, `make`, `mkdir`, `mkfs`, `mknod`, `mount`, `mv`, `od`, `passwd`, `pr`, `pwd`, `rev`, `rm`, `rmdir`, `roff`, `sh`, `size`, `sleep`, `sort`, `split`, `stty`, `su`, `sum`, `sync`, `tail`, `tar`, `tee`, `time`, `touch`, `tr`, `umount`, `uniq`, `update`, and `wc`. A full-screen editor inspired by Emacs (think of it as nano-emacs), a full K&R compatible C compiler, and programs to read and write MS-DOS diskettes are also included. All of the sources of the operating system and these utilities, except the C compiler source (which is quite large and is available separately), are included in the software package.

In addition to the above utilities, over 100 library procedures, including `stdio`, are provided, again with the full source code.

To reiterate what was said above, all of this software is completely new. Not a single line of it is taken from, or even based on, the AT&T code. I personally wrote from scratch the entire operating system and some of the utilities. This took about 3 years. My students and some other generous people wrote the rest. The C compiler is derived from the Amsterdam Compiler Kit (Tanenbaum et al., 1983), and was written at the Vrije Universiteit. It is a top-down, recursive descent compiler written in a compiler writing language called LLGEN and is not related to the AT&T portable C compiler, which is a bottom-up, LALR compiler written in YACC.

2. Overview of the MINIX System Architecture

UNIX is organized as a single executable program that is loaded into memory at system boot time and then run. MINIX is structured in a much more modular way, as a collection of processes that communicate with each other and with user processes by sending and receiving messages. There are separate processes for the memory manager, the file system, for each device driver, and for certain other system functions. This structure enforces a better interface between the pieces. The file system cannot, for example, accidentally change the memory manager's tables because the file system and memory manager each have their own private address spaces.

These system processes are each full-fledged processes, with their own memory allocation, process table entry and state. They can be run, blocked, and send messages, just as the user processes. In fact, the memory manager and file system each run in user space as ordinary processes. The device drivers are all linked together with the kernel into the same binary program, but they communicate with each other and with the other processes by message passing.

When the system is compiled, four binary programs are independently created: the kernel (including the driver processes), the memory manager, the file system, and `init` (which reads `/etc/ttys` and forks off the login processes). In other words, compiling the system results in four distinct `a.out` files. When the system is booted, all four of these are read into memory from the boot diskette.

It is possible, and in fact, normal, to modify, recompile, and relink, say, the file system, without having to relink the other three pieces. This design provides a high degree of modularity by dividing the system up into independent pieces, each with a well-defined function and interface to the other pieces. The pieces communicate by sending and receiving messages.

The various processes are structured in four layers:

4. The user processes (top layer).
3. The server processes (memory manager and file system).
2. The device drivers, one process per device.
1. Process and message handling (bottom layer).

Let us now briefly summarize the function of each layer.

Layer 1 is concerned with doing process management including CPU scheduling and interprocess communication. When a process does a SEND or RECEIVE, it traps to the kernel, which then tries to execute the command. If the command cannot be executed (e.g., a process does a RECEIVE and there are no messages waiting for it), the caller is blocked until the command can be executed, at which time the process is reactivated. When an interrupt occurs, layer 1 converts it into a message to the appropriate device driver, which will normally be blocked waiting for it. The decision about which process to run when is also made in layer 1. A priority algorithm is used, giving device drivers higher priority over ordinary user processes, for example.

Layer 2 contains the device drivers, one process per major device. These processes are part of the kernel's address space because they must run in kernel mode to access I/O device registers and execute I/O instructions. Although the IBM PC does not have user mode/kernel mode, most other machines do, so this decision has been made with an eye toward the future. To distinguish the processes within the kernel from those in user space, the kernel processes are called *tasks*.

Layer 3 contains only two processes, the memory manager and the file system. They are both structured as *servers*, with the user processes as *clients*. When a user process (i.e. a client) wants to execute a system call, it calls, for example, the library procedure `read` with the file descriptor, buffer, and count. The library procedure builds a message containing the system call number and the parameters and sends it to the file system. The client then blocks waiting for a reply. When the file system receives the message, it carries it out and sends back a reply containing the number of bytes read or the error code. The library procedure gets the reply and returns the result to the caller in the usual way. The user is completely unaware of what is going on here, making it easy to replace the local file system with a remote one.

Layer 4 contains the user programs. When the system comes up, `init` forks off login processes, which then wait for input. On a successful login, the shell is executed. Processes can fork, resulting in a tree of processes, with `init` at the root. When CTRL-D is typed to a shell, it exits, and `init` replaces the shell with another login process.

3. Layer 1 — Processes and Messages

The two basic concepts on which MINIX is built are processes and messages. A process is an independently schedulable entity with its own process table entry. A message is a structure containing the sender's process number, a message type field, and a variable part (a union) containing the parameters or reply codes of the message. Message size is fixed, depending on how big the union happens to be on the machine in question. On the IBM PC it is 24 bytes.

Three kernel calls are provided:

- RECEIVE(source,&message)
- SEND(destination,&message)
- SENDREC(process,&message)

These are the only true system calls (i.e. traps to the kernel). RECEIVE announces the willingness of the caller to accept a message from a specified process, or ANY, if the RECEIVER will accept any message. (From here on, "process" also includes the tasks.) If no message is available, the receiving process is blocked. SEND attempts to transmit a message to the destination process. If the destination process is currently blocked trying to receive from the sender, the kernel copies the message from the sender's buffer to the receiver's buffer, and then marks them both as runnable. If the receiver is not waiting for a message from the sender, the sender is blocked.

The SENDREC primitive combines the functions of the other two. It sends a message to the indicated process, and then blocks until a reply has been received. The reply overwrites the original message. User processes use SENDREC to execute system calls by sending messages to the servers and then blocking until the reply arrives.

There are two ways to enter the kernel. One way is by the trap resulting from a process' attempt to send or receive a message. The other way is by an interrupt. When an interrupt occurs, the registers and machine state of the currently running process are saved in its process table entry. Then a general interrupt handler is called with the interrupt number as parameter. This procedure builds a message of type INTERRUPT, copies it to the buffer of the waiting task, marks that task as runnable (unblocked), and then calls the scheduler to see who to run next.

The scheduler maintains three queues, corresponding to layers 2, 3, and 4, respectively. The driver queue has the highest priority, the server queue has middle priority, and the user queue has lowest priority. The scheduling algorithm is simple: find the highest priority queue that has at least one process on it, and run the first process on that queue. In this way, a clock interrupt will cause a process switch if the file system was running, but not if the disk driver was running. If the disk driver was running, the clock task will be put at the end of the highest priority queue, and run when its turn comes.

In addition to this rule, once every 100 msec, the clock task checks to see if the current process is a user process that has been running for at least 100 msec. If so, that user is removed from the front of the user queue and put on the back. In effect, compute bound user processes are run using a round robin scheduler. Once started, a user process runs until either it blocks trying to send or receive a message, or it has had 100 msec of CPU time. This algorithm is simple, fair, and easy to implement.

4. Layer 2 — Device Drivers

Like all versions of UNIX for the IBM PC, MINIX does not use the ROM BIOS for input or output because the BIOS does not support interrupts. Suppose a user forks off a compilation in the background and then calls the editor. If the editor tried to read from the terminal using the BIOS, the compilation (and any other background jobs such as printing) would be stopped dead in their tracks waiting for the the next character to be typed. Such behaviour may be acceptable in the MS-DOS world, but it certainly is not in the UNIX world. As a result, MINIX contains a complete set of drivers that duplicate the functions of the BIOS. Like the rest of MINIX, these drivers are written in C, not assembly language.

This design has important implications for running MINIX on PC clones. A clone whose hardware is not compatible with the PC down to the chip level, but which tries to hide the differences by making the BIOS calls functionally identical to IBM's will not run an unmodified MINIX because MINIX does not use the BIOS.

Each device driver is a separate process in MINIX. At present, the drivers include the clock driver, terminal driver, various disk drivers (e.g., RAM disk, floppy disk), and printer driver. Each driver has a main loop consisting of three actions:

1. Wait for an incoming message.
2. Perform the request contained in the message.
3. Send a reply message.

Request messages have a standard format, containing the opcode (e.g., READ, WRITE, or IOCTL), the minor device number, the position (e.g., disk block number), the buffer address, the byte count, and the number of the process on whose behalf the work is being done.

As an example of where device drivers fit in, consider what happens when a user wants to read from a file. The user sends a message to the file system. If the file system has the needed data in its buffer cache, they are copied back to the user. Otherwise, the file system sends a message to the disk task requesting that the block be read into a buffer within the file system's address space (in its cache). Users may not send messages to the tasks directly. Only the servers may do this.

MINIX supports a RAM disk. In fact, the RAM disk is always used to hold the root device. When the system is booted, after the operating system has been loaded, the user is instructed to insert the root file system diskette. The file system then sees how big it is, allocates the necessary memory, and copies the diskette to the RAM disk. Other file systems can then be mounted on the root device.

This organization puts important directories such as `/bin` and `/tmp` on the fastest device, and also makes it easy to work with either floppy disks or hard disks or a mixture of the two by mounting them on `/usr` or `/user` or elsewhere. In any event, the root device is always in the same place.

In the standard distribution, the RAM disk is about 240K, most of which is full of parts of the C compiler. In the 256K system, a much smaller RAM disk has to be used, which explains why this version has no C compiler: there is no place to put it. (The `/usr` diskette is completely full with the other utility programs and one of the design goals was to make the system run on a 256K PC with 1 floppy disk.) Users with an unusual configuration such as 256K and three hard disks are free to juggle things around as they see fit.

The terminal driver is compatible with the standard V7 terminal driver. It supports cooked mode, raw mode, and cbreak mode. It also supports several escape sequences, such as cursor positioning and reverse scrolling because the screen editor needs them.

The printer driver copies its input to the printer character for character without modification. It does not even convert line feed to carriage return + line feed. This makes it possible to send escape sequences to graphics printers without the driver messing things up. MINIX does not spool output because floppy disk systems rarely have enough spare disk space for the spooling directory. Instead one normally would print a file `f` by saying

```
lpr <f &
```

to do the printing in the background. The `lpr` program inserts carriage returns, expands tabs, and so on, so it should only be used for straight ASCII files. On hard disk systems, a spooler would not be difficult to write.

5. Layer 3 — Servers

Layer 3 contains two server processes: the memory manager and the file system. They are both structured in the same way as the device drivers, that is a main loop that accepts requests, performs them, and then replies. We will now look at each of these in turn.

The memory manager's job is to handle those system calls that affect memory allocation, as well as a few others. These include `FORK`, `EXEC`, `WAIT`, `KILL`, and `BRK`. The memory model used by MINIX is exceptionally simple in order to accommodate computers without any memory management hardware. When the shell forks off a process, a copy of the shell is made in memory. When the child does an `EXEC`, the new core image is placed in memory. Thereafter it is never moved. MINIX does not swap or page.

The amount of memory allocated to the process is determined by a field in the header of the executable file. A program, `chmem`, has been provided to manipulate this field. When a process is started, the text segment is set at the very bottom of the allocated memory area, followed by the data and `bss`. The stack starts at the top of the allocated memory and grows downward. The space between the bottom of the stack and the top of the data segment is available for both segments to grow into as needed. If the two segments meet, the process is killed.

In the past, before paging was invented, all memory allocation schemes worked like this. In the future, when even small microcomputers will use 32-bit CPUs and $1\text{M} \times 1$ bit memory chips, the minimum feasible memory will be 4 megabytes and this allocation scheme will probably become popular again due to its inherent simplicity. Thus the MINIX scheme can be regarded as either hopelessly outdated or amazingly futuristic, as you prefer.

The memory manager keeps track of memory using a list of holes. When new memory is needed, either for `FORK` or for `EXEC`, it searches the hole list and takes the first hole that is big enough (first fit). When a process terminates, if it is adjacent to a hole on either side, the process' memory and the hole are merged into a bigger hole.

The file system is really a remote file server that happens to be running on the user's machine. However it is straightforward to convert it into a true network file server. All that needs to be done is change the message interface and provide some way of authenticating requests. (In MINIX, the source field in the incoming message is trustworthy because it is filled in by the kernel.) When running remote, the MINIX file server maintains state information, like RFS and unlike NFS.

The MINIX file system is similar to that of V7 UNIX. The i-node is slightly different, containing only 9 disk addresses instead of 13, and only 1 time instead of 3. These changes reduce the i-node from 64 bytes to 32 bytes, to store more i-nodes per disk block and reduce the size of the in-core i-node table.

Free disk blocks and free inodes are kept track of using bit maps rather than free lists. The bit maps for the root device and all mounted file systems are kept in memory. When a file grows, the system makes a definite effort to allocate the new block as close as possible to the old ones, to minimize arm motion. Disk storage is not necessarily allocated one block at a time. A minor device can be configured to allocate 2, 4 (or more) contiguous blocks whenever a block is allocated. Although

this wastes disk space, these multiblock *zones* improve disk performance by keeping file blocks close together. The standard parameters for MINIX as distributed are 1K blocks and 1K zones (i.e. just 1 block per zone).

MINIX maintains a buffer cache of recently used blocks. A hashing algorithm is used to look up blocks in the cache. When an i-node block, directory block, or other critical block is modified, it is written back to disk immediately. Data blocks are only written back at the next SYNC or when the buffer is needed for something else.

The MINIX directory system and format is identical to that of V7 UNIX. File names are strings of up to 14 characters, and directories can be arbitrarily long.

6. *Layer 4 — User Processes*

This layer contains *init*, the shell, the editor, the compiler, the utilities, and all the user processes. These processes may only send messages to the memory manager and the file system, and these servers only accept valid system call requests. Thus the user processes do not perceive MINIX to be a general-purpose message passing system. However, removing the one line of code that checks if the message destination is valid would convert it into a much more general system (but less UNIX-like).

7. *Documentation*

For a system one of whose purposes is teaching about operating systems, ample documentation is essential. For this reason I have written an ample textbook (more than 700 pages) treating both the theory and the practice of operating system design (Tanenbaum, 1987). The table of contents is as follows:

Chapters

1. Introduction
2. Processes
3. Input/Output
4. Memory Management
5. File Systems
6. Bibliography and Suggested Readings

Appendices

- A. Introduction to C
- B. Introduction to the IBM PC
- C. MINIX Users Guide
- D. M INIX Implementers Guide
- E. MINIX Source Code Listing
- F. MINIX Cross Reference Map

The heart of the book is chapters 2 — 5. Each chapter deals with the indicated topic in the following way. First comes a thorough treatment of the relevant principles (thorough enough to be usable as a university textbook on operating systems). Next comes a general discussion of how the principles have been applied in MINIX. Finally there is a procedure by procedure description of how the relevant part of MINIX works in detail. The source code listing of Appendix E

contains line numbers, and these line numbers are used throughout the book to pinpoint the code under discussion. The source code itself contains more than 3000 comments, some more than a page long. Studying the principles and seeing how they are applied in a real system gives the reader a better understanding of the subject than either the principles or the code alone would.

Appendices A and B are quickie introductions to C and the IBM PC for readers not familiar with these subjects. Appendix C tells how to boot MINIX, how to use it, and how to shut it down. It also contains all the manual pages for the utility programs. Most important of all, it gives the super-user password.

Appendix D is for people who wish to modify and recompile MINIX. It contains a wealth of nutsy-boltsy information about everything from how to use MS-DOS as a development system, to what to do when your newly made system refuses to boot.

Appendix E is a full listing of the operating system, all 260 pages of it. The utilities (mercifully) are not listed.

8. *Distribution of the Software*

The software distribution is being done by Prentice-Hall. Four packages are available. All four contain the full source code; they differ only in the configuration of the binary supplied. The four packages are:

- 640K IBM PC version
- 256K IBM PC (no C compiler)
- IBM PC-AT (512K minimum)
- Industry standard 9-track tape

The 640K version will also run on 512K systems, but it may be necessary to chmem parts of the C compiler to make it fit. The tape version is the only one containing the IBM PC simulator and other software needed for classroom use on a VAX or other time sharing machine. The software packages do not include the book.

If there is sufficient interest, a newsgroup net.minix will be set up. This channel can be used by people wishing to contribute new programs, point out and correct bugs, discuss the problems of porting MINIX to new systems, etc.

9. *Acknowledgements*

I would like to thank the following people for contributing utility programs and advice to the MINIX effort: Martin Atkins, Erik Baalbergen, Charles Forsyth, Richard Gregg, Michiel Huisjes, Patrick van Kleef, Adri Koppes, Paul Ogilvie, Paul Polderman, and Robbert van Renesse. Without their help, the system would have been far less useful than it now is.

10. *References*

Tanenbaum, A.S., van Staveren, H., Keizer, E.G., and Stevenson, J.W.: *A Practical Toolkit for Making Portable Compilers*, Communications of the ACM, Vol. 26, pp. 654-660, September 1983.

Tanenbaum, A.S.: *Operating Systems: Design and Implementation*, Englewood Cliffs, N.J.: Prentice-Hall, 1987.

A User Programmable Telephone Switch

Brian E. Redman

*Bell Communications Research
Morristown, New Jersey 07960, USA*

The basic function of a telephone switch is to allow subscribers to place calls among one another. The basic service provided is Plain Old Telephone Service (POTS). There were relatively few changes in POTS since telephone switching was introduced in 1880. In 1919 dial service became available alleviating the need for operator assistance on many calls. Direct Distance Dialing (DDD) in 1951 expanded this to long distance calls. In 1964 touch-tone service provided faster and easier dialing. It wasn't until 1972 that Vertical Services were offered to residence costumers. These were services such as Speed Calling, Call Waiting and Call Forwarding.

When you rented a pair of telephones in 1887 there was only one option available. For an additional \$5 installation charge they were equipped with thumpers, the predecessor of the bell. Otherwise you could simply yell into your telephone and hope the other party was close enough to theirs to hear you. When you subscribe to telephone service today you are offered a number of enhancements to POTS. Unfortunately the precise behavior and control of these options is quite limited.

Nowadays telephone switching systems are controlled by computers. There is the capacity to do more than switch calls among subscribers. It is both practical and attractive to have telephone services controlled dynamically by the subscriber, either via direct input to the telephone switching system or by exercising customer designed control algorithms.

The telephone switching system which will be described provides several interfaces to the subscriber. At the highest levels, the user can activate or deactivate preprogrammed algorithms and modify their behavior to the extent that such modification has been allowed for in their design. This is achieved with touch-tone input or by issuing commands from a computer terminal. At the intermediate level the user can incorporate program library functions and implement control algorithms with their own computer programs. At the lowest level users can claim total control of their assigned circuits.

This system has been in use for over a year, providing essential telephone service to twenty subscribers. Although the basic design has remained intact, the emphasis on utilization of the different layers is shifting markedly.

1. Introduction

The BerBell user programmable telephone switch places into the hands of its customers the responsibility of determining how their telephone should behave

beyond a basic standard service. There are so many options available in modern systems that it is not reasonable for their designers or installers to predict or restrict the desires of each individual. By providing a powerful set of primitives and a clean interface, the subscribers themselves or their agents can dictate the functionality of their service. Thus service definition is open-ended, evolving with the needs and imagination of the system's users. The exploitation of BerBell's capabilities has resulted in a continually growing library of user programs and services. These include placing calls from an on-line directory and scheduling calls from an on-line calendar. Users with computer access can take somewhat greater advantage of BerBell. Whenever possible, the telephone touch-tone interface provides similar capabilities to the computer terminal interface. However more complex features are more easily manipulated with the use of textual input and output. Although rotary dialing cannot be fully supported, BerBell will function well with all types of touch-tone telephones and computer terminals. Putting the data bases and features within the system or within reach of the system through computer networks obviates the need for expensive special purpose accessories.

The work described involves the use of a general purpose operating system, UNIX, and its associated program resources to support the development and application of a telephone switch. The system as a whole is referred to as BerBell. Its software consists of a core program, *bellerophon*, a number of programs varying in their degree of independence from *bellerophon*, the UNIX operating system and its tools. The hardware which realises the system is composed of a host minicomputer, a Redcom Modular Switching Peripheral providing the basic electrical interfaces and switching capabilities required for telephony applications, speech synthesizers and an assortment of ancillary audio sources and recorders. These components together provide flexible and comprehensive telephone service.

2. User Level

How is BerBell different from other telephone offerings from the end-user's point of view? What new capabilities are there? There are no significant differences between the basic service provided by BerBell and that provided by other vendors. The default BerBell dialing plan is designed to look like CENTREX since most subscribers use it at work. However, dialing plans are associated with the telephone line, so home subscribers can use the standard residence dialing plan. Basic service implies that when the subscriber lifts the receiver, they hear dialtone. They then dial a valid number and a call is placed to their party. On the receiving side, if the telephone is in use callers get a busy tone. Otherwise the telephone is rung. If the receiving party answers, a talking connection is established.

BerBell and most other vendors provide more interesting services upon request. BerBell subscribers can activate and disconnect these features using a touch-tone telephone or by issuing shell level commands from a computer served by the BerBell host. Although the fundamental concepts of these features are similar, BerBell advances their applications. First, we present a discussion of those features that are typically provided by most vendors.

2.1 Call Forwarding

Subscribers can arrange to have their calls transferred to another number. Typical residence service only allows unconditional call forwarding. Most PBX and CENTREX vendors provide call forwarding when the called line is busy as well as after the line has rung some number of times (no answer). These functions are available to BerBell subscribers with some improvements. Most importantly, the parameters of each of the call forwarding operations are conveniently changeable by the subscriber.

From a touch-tone telephone, the user enters "*" (non-call dialing), then "2" indicating a feature setting, followed by "1" (call forwarding) then various codes to set parameters. The feature dialing syntax is designed to be consistent and hierarchical. It is simpler to use the computer interface to describe these parameters. In all cases the command is

```
setforward <extension> <option>.
```

The basic call forwarding options are:

rings <n>

the number of rings after which no-answer forwarding will be effected. If <n> is zero unconditional forwarding is activated.

busy/nobusy

activate/deactivate forwarding when the line is busy.

noanswernumber <number>

the number to transfer the call to if no-answer forwarding is active and the call is not answered after the specified number of rings. The name of a program can be substituted for <number>.

unconditionalnumber <number>

the number or program to transfer the call to if rings is set to zero.

busynumber <number>

the number or program to transfer the call to if the line is busy and busy forwarding is activated.

Some new call forwarding operations are implemented to allow callers and recipients of forwarded calls to be made aware that call forwarding has been invoked. The terms "inform" and "announce" are used to indicate messages to the caller and ultimate recipient respectively. The following options involve speaking a text message synthetically or playing a recorded message.

inform/noinform

enable/disable calling party notification that the call is being forwarded.

announce/noannounce

enable/disable recipient party notification that the call has been forwarded to them.

For each of the forwarding conditions described a different message can be specified with <file> which contains text to be recited or binary audio data.

busyannounce <file>

noanswerannounce <file>

unconditionalannounce <file>

busyinform <file>

noanswerinform <file>

unconditionalinform <file>

A "continueringing" option allows the originally called line to continue ringing even after it has been forwarded. It can then be answered at any time. This is disabled with "stopringing".

Finally, the "on/off" option will activate/deactivate all forwarding without modifying the parameter settings described above.

2.2 Call Waiting

A call to a party whose line is busy causes audible ringing to be heard by the caller and a short tone by the recipient. The recipient may then talk to the calling party by hookflashing, placing the current conversation on hold. In the BerBell implementation call waiting can be enabled or disabled from the telephone by pressing "*20" then "1" or "0" respectively. From a terminal the command

```
setcw [on | off] <program name>
```

is used to specify a program which is executed when the caller encounters a busy line with call waiting enabled. As in call forwarding and most other services the program can be supplied by the user. A popular program in use for this purpose informs the caller that their party will be responding shortly, then connects them to silence, music or an answering program, at their option. The recipient, having heard a high-pitched tone when the new call arrived will hear a low-pitched tone if the new caller should hang up. In fact the new call is simply a held call and the subscriber will hear a low-pitched tone any time a held caller hangs up. Many calls can be on hold simultaneously.

2.3 Call Transfer

A call in progress can be forwarded to another number. This is a fairly common feature but BerBell provides a slight twist. In the normal case a call is transferred using the telephone by first placing it on hold, then dialing "*12", then the slot number or "#" for the oldest held call, then the destination number. From the terminal the user issues

```
xfer <extension> <destination number>.
```

In this case the call in progress, not a held call, is transferred. The twist mentioned is that calls can be temporarily transferred. That is to say that the call is transferred but still remains on hold. This means that the user can still pick up the call, etc. This feature conveniently implements services on hold such as music, advertising, games, or information. Each of these services is implemented as a program which can be designed by the subscriber. Programs exist which give the caller the option to select a preference (including silence). In any event the options are dictated by the subscriber and the choice is made by the user, not the system. To invoke the temporary transfer from a telephone, place the call on hold then dial "*16#", then the number to call. The command to issue from a terminal is

```
txfer <extension> <number>.
```

Like `xfer` the current call is transferred, not a held call.

The more common features found in most telephone systems have been covered. The discussion now turns to some less common services.

2.4 Editing

BerBell incorporates some editing facilities similar to those used for computer terminal interfaces. These are "*##", erase the last keypress; "***#" erase all dialed digits; and "***" recite the digits dialed so far.

2.5 Call Announcement

A program can be executed when a subscriber receives a call. The command

```
setcallfor [on | off] <program>
```

is used to control the option. From the telephone, "*221" and "*220" are used to activate or deactivate the feature. The default program utilises a switched public

address system to announce to subscribers that their telephones are ringing. The user can provide a text or data file to be spoken by a speech synthesizer or played by a digital sound device. Another file specifies at which locations the announcement is to be made. Users can answer their calls from any BerBell extension.

2.6 Additional Manipulations of Calls

It is possible to pick up a call that is ringing on another telephone, or to redirect it to another number. Other functions can be performed on held calls. By dialing `**14#` the user invokes "hold-on-hold". This program, dubbed "revenge on hold" and designed on a whim, causes the held call to repetitively receive a message advising the party to dial `**` when they return to the telephone. The user's line is freed for incoming or outgoing calls. When the held party does dial `**` the user's telephone is rung just as if they were receiving a call, and upon answering they are connected to their party.

Two additional features which apply to held calls are held retrieval and held transfer. Held transfer allows a user to transfer a held call to another extension while in the held state. Thus the call appears, still on hold, on another extension. The code is `**15#`, then the destination extension. Held retrieval permits a user to pick up a call which is on hold on another extension. The code for this is `**18#`, then the extension from which the held call will be retrieved.

2.7 Programs

User programs provide a large portion of BerBell's functionality. The system is designed specifically to give the programmer flexibility and control over the telephone switch. Users issue commands from the BerBell host or remotely from any machine that communicates with the host over the Internet. In the 4.3 BSD implementation, which uses Internet Datagrams to communicate among processes within BerBell, the programs that are used locally function identically over the network. What follows is a brief description of some of the programs in common use now. The list is not exhaustive and continues to grow. One should also note that many of the functions described above will be recoded as stand alone programs.

2.7.1 Accessed from the Telephone

The following list of programs are services invoked by *bellerophon* as a subscriber option or a general service.

- `bebap` The default answering program. It allows callers to leave a number or a message, or connect to a message taking persons. Callers can also receive messages that are left for them.
- `ttda` Touch-tone directory assistance^[1] permits users to look up telephone numbers in various telephone books. The user can be transferred to the matched number. Current directories include Bellcore, Seattle, and many New Jersey books.
- `ttweather`
Using touch-tone input, the user gets the National Weather Service forecast for any desired city in the continental U.S.
- `ttsh` Using a touch-tone mapping scheme where two keypresses represent one ASCII character, a user can log in to the BerBell host and execute commands.
- `demo2332`
The audio research laboratory real-time music generation demonstration.
- `wakeup`
The wakeup service calls the user at a desired time, and delivers a message.

ccstatus

Computer Center Status provides information to the caller and permits them to enter their number to receive updates. When staff personnel change the status file, users are called back.

wrong The wrong-number server randomly executes one of many BERPS scripts to entertain and confuse callers who dial invalid BerBell extensions. BERPS is described in a subsequent section. These scripts include simulations of telephone interfaces to various institutions such as banks, the Federal Government, the Phone Company, the Defense Department and the Underworld.

robop The robot operator recites the list of dial-up services.

help An interactive program for feature dialing assistance.

htime Recites the local time, date, and weather.

musak Connects the caller to an arbitrary audio source.

ph/silence

These services are primarily for testing, connecting the caller to a permanent high-pitched tone or to silence.

hanna Simulates the utterances of a three year old child.

newsgen

Assembles an inane scandal sheet news story from a table of phrases and personalities.

rosary Recites same.

winner Announces to the caller that they have won something and asks them to hold on (indefinitely).

marge Simulates the gratuitous utterances of a cashier.

suicide

An irreverent suicide hot-line.

The following programs are issued from the user's terminal.

call <from> <to>

The <from> number is called. When it is answered, a call is placed from it to <to> number.

nway <number> <number> <number> [number] ...

Conference calls are established with this command.

gm [options] <from> <to> [message file]

Getme dials the <to> number and may deliver the optional message contained in *message file*. *options* may be used to elicit a response from the answerer at <to> number. If a satisfactory response is received or if no response is required, the call is connected to <from> number.

pa <number> <message file>

<number> is dialed and the contents of <messagefile> is delivered.

3. *The Programmers View*

There are several programming interfaces to BerBell. For implementing user level functions such as the programs described above, there are C language library routines, the BERPS interpretive language and the UNIX shell. Programs may be

invoked implicitly by a user or by the system or explicitly by typing at a computer terminal. The system will invoke a program when an inbound call to a number associated with that program is received (e.g., directory assistance), when a feature implicates a program (e.g., recite speed codes) or when a user's feature settings dictate (e.g., call waiting). In each of these cases the arguments to the program will include the name of the circuit holding the call and the number dialed. Programs can then connect auxiliary devices such as tones, recorded audio, speech synthesizers, answering machines and audio components to the call, reroute the call, hold or release it.

3.1 C Programming

There are C libraries for manipulating BerBell objects, speech synthesizers, touch-tone receivers and digital audio input and output channels. The BerBell library contains six "system calls" that allow the programmer to easily connect objects such as trunks and lines* among one another and to place telephone calls using these objects. It is straightforward to write C programs using this library. No significant knowledge of telephony is required to take substantial control over your communications channels.

3.2 BERPS interpreter

BERPS stands for BER Phone Script. It is a simple language that interprets scripts which manipulate telephones. It is designed to alleviate the need for users to program in C in order to write application programs. The wrong-number servers are written in BERPS as well as an answering, call waiting, and call announcement program.

The interpreter itself was written using the library functions. The language provides for flow control, arithmetic calculations, BerBell operations and operating system services. The following example implements one of the wrong-number services.

```
# Trans Galactic Phone Network
  [a comment.]
%h z
  [jump to label z (:z) when the caller hangs up.]
You have connected to the intergalactic communications network.
Please enter galaxy code, use sharp or pound to terminate.
%r g
  [read a number input by the user into the variable "g".]
Now enter planet destination.
%r p
%h l
%= p 10
  [execute commands until the "%" if the variable "p" has the value "10".]
Planet Ten is in the Eighth Dimension.
Please dial the interdimension operator for assistance.
%l p 10
  [the "not equal" case.]
```

* Trunks connect telephone switches together, lines connect telephone instruments to switches. Within BerBell, audio equipment is connected to the switch by trunk interfaces.

```

We are sorry, planet $Np in galaxy $Ng has been obliterated.
Please check the code and call again.
%}
:l
%F /logs/wrong $T: galaxy $ng planet $np ($P)
    [append to file.]
%e 0
    [exit.]
:z
%F /logs/wrong $T: caller hung up ($P)

```

Text is simply spoken through a speech synthesizer. \$Np expands the value of p to text as digits separated by spaces. \$n expands a variable as a single number. \$T is the current time, \$P is the process identifier.

3.3 Shell Programming

The interface between user processes and the core BerBell process uses pseudo-ttys on the Eighth Edition version and Internet Datagrams on the 4.3 BSD implementation. Thus one may simply echo commands to a named pseudo-tty or use a special echo which deals with Datagrams as required. As usual, examples of Shell code look awful.

4. Operators

At the dawn of the telephone business, boys were employed as switchboard operators. But,

Boys did not last very long as operators. At the time they were often impatient, rude, and foulmouthed to the subscribers^[2].

Another set of C library functions exists for programming BerBell operations at a lower level. Programs written at this "supervisory" level are termed "operators" and are responsible for complete control of individual circuits in cooperation with the BerBell kernel program. While user level programs are unaware of system details, operators require some knowledge of the switching conventions and the hardware functionality. A program using these functions would receive hardware state changes from and issue commands directly to the circuit or circuits under its control over IPC channels.

It is likely that all call processing will be done by autonomous operator programs distributed over several machines. The advantages are clear in terms of processor utilization, flexibility, customization and prototyping. The operator concept has proven useful and rewarding, allowing a quick, permanent and elegant solution to several unforeseen problems.

5. Software Descriptions

The BerBell software is made up of a number of permanently executing programs and processes that communicate through pipes and pseudo-ttys or Internet sockets. During the course of operation new short lived programs are invoked that implement specific features and services as described previously.

5.1 MSP Protocol Program

The switch used is called a Modular Switching Peripheral (MSP). It is described in detail in the Hardware section. The communications between the software system and the hardware switch is done over an RS232 serial port using the ANSI X3.28

protocol. This program was written at Bellcore in Navesink and trivial changes were made locally to convert it from UNIX System V to the Eighth Edition and 4.3 BSD. The program consist of four concurrently executed modules. The first is a parent program which opens the serial line and sets the appropriate options such as baud rate, parity, etc. It then sets up communication pipes and invokes the "control", "host" and "msp" processes. Paraphrasing from the commented C program:

The *control* process controls both the *host* and *msp* processes. It reads from its input and changes state and takes actions depending on the present state and the input. This process takes care of all procedural messages for granting master/slave status.

The *host* process reads a string from its standard input to be sent to the device using the ANSI protocol. This process notifies the *control* process for permission to transmit. The *control* process notifies the *host* process when it is ready, and the *host* process transmits the message. The *host* process informs the *control* process regarding the success of the transfer.

The *msp* process reads from the MSP and forwards any characters to the *control* process. It waits to receive a response from the *control* process before continuing.

The MSP protocol program is invoked from the core switching process.

5.2 The Core Program

The main, once monolithic, program is named "bellerophon". It sets up communication pipes and invokes a filtering program that processes all the textual output, a status program to which hardware state changes are sent, the protocol program as described above, and a DECTalk server program. These will be described in succeeding sections. Communication with these processes is implemented using operating system dependent macros. In the case of Eighth Edition a pseudo-tty is opened by *bellerophon*, the device name is linked to a name known to other programs thus implementing a "named pipe". Under 4.3 BSD a named socket is opened which receives Datagrams. These provide the input channel to *bellerophon*.

Bellerophon then initialises the hardware and data structures. There is a structure for each port and for each telephone number. The status process mentioned previously maintains the state of each port as represented in an internal data structure in a file. This file is read by an initialization routine to determine if the port ought to be initialised. It will not be initialised if it is not idle, thus calls in progress during a software reboot are not affected.

Bellerophon then enters its main loop gathering data from the MSP or from its input channel. Input from the MSP is parsed to identify the port involved. The current state is used to determine the function to be called. This function receives the port's data structure and any other data provided by the MSP (e.g., digits received) as arguments. In the case of data from the input channel, a function is called which acts on the input, a command and its arguments. A status reply is sent by writing to the device (or file) named in the arguments. Under Eighth Edition it is typically a pseudo-tty. In the 4.3 BSD implementation the recipient's address is part of the received message.

5.3 State Information Program

The program *statproc* maintains a file with the current state of each circuit. Each state change sends the circuit name and new state through a pipe to *statproco*. Other reported data includes dialed digits and circuit connection. As well as

maintaining the state file, *statproc* also disseminates state information to other programs. By reading commands on its input stream in a similar manner to that in which *bellerophon* accepts its commands, *statproc* is informed of the appearance of clients who wish to receive status updates. Clients include programs that display system activity on bitmapped terminals, monitor the system's heartbeat (reported every three seconds), and maintain usage statistics in real time.

6. Hardware

The following sections describe the current hardware compliment, not the minimal or ideal configuration. Figure 1. is a schematic representation of the major hardware components.

6.1 Redcom Switches

The Modular Switching Peripheral (MSP) from Redcom Labs has proved a satisfactory piece of hardware for experimental applications. It provides the electrical interfaces required for telephone switching and does not interfere with the programmer's need to control its functionality.

6.2 Host Computers

For the sake of experimental diversity, two different system configurations are implemented.

6.2.1 VAX 11/750

The first system was implemented on the VAX 11/750 running UNIX Eighth Edition. This system supports approximately five full-time users with a general time sharing environment as well as the experimental telephone switch application. The VAX is configured with 4 megabytes of memory and 1.4 gigabytes of disk storage on 3 ra81s and 1 ra60. There are four DZ11s providing 32 serial lines. Thirteen of these lines are dedicated to BerBell, interfacing DECTalks, the Cytek switch, and the MSPs. The VAX is also equipped with the DSC-200 connected to the UNIBUS, an Interlan Ethernet controller, and miscellaneous other peripherals.

6.2.2 MicroVax II

The MicroVax is a relatively new addition to the experimental telephone laboratory. The operating system is 4.3 BSD. It is equipped with four megabytes of main memory and 140 megabytes of disk storage. There are 8 serial lines and an Ethernet controller. It is connected to the single shelf experimental switch which is dependent upon the VAX BerBell system for access the DDD network.

6.3 Audio Devices

There are 11 DECTalks in use, 8 DTC03s and 3 DTC01s. The DTC03 is rack mounted and functionally superior to the DTC01 stand-alone models. The main functional improvement is the ability of the DTC03 to automatically terminate speech when a user presses a button on the telephone keypad. Providing this needed function in the host's software on the DTC01s was a somewhat exasperating exercise. DTC01s are still in use because they alone are equipped with terminal interfaces and audio output separate from the telephone interface. They are also more appropriate for touch-tone signaling of other devices such as the Watson.

Recording voice messages is an absolute necessity. Users expect at least the capabilities of a conventional answering machine in their sophisticated telephone environment. The system utilises such answering machines principally as backup devices as well as the Watson, a sort of programmable multiuser answering machine and the DSC 200, a truly programmable audio record and playback unit.

6.4 Audio

There is an experimental audio lab accessible by BerBell. BerBell serves to provide access from the DDD network to this lab in order to demonstrate ongoing audio/music research^[3]. It consists of a MIDI controlled studio of synthesizers and percussion machines. The MIDI host is a SUN 3/160.

There are also a number of other audio program sources attached to BerBell to give users a choice of entertainment on hold or otherwise.

7. Evolution

In April of 1985 an RS232-controlled telephone switch was obtained. The switch, manufactured by Redcom Laboratories, is described in detail in the Hardware section of this document. With the switch came two pieces of software: an ANSI X3.28 protocol handling program to support the low level communications between the switch and the host was written by colleagues at the Navesink Research and Engineering Center of Bellcore and remains in use today. A rudimentary call processing program, provided by the Network Architecture Research Division at Morristown, was useful in bootstrapping the system.

The initial application for the switch met two requirements. The first was to provide new and interesting services such as "a better answering machine"^[4]. The second was to reimplement services found in modern switches, but with enhancements making them more useful.

BerBell grew naturally and easily from the desire to provide better telephone service for an individual user to its current power and generality through circumstance and experience as well as vision. The requirement to switch telephone calls was initially fulfilled with a homemade switch consisting of a matrix of relays and a UART. The author originally built this device to permit several computer terminals to share half as many computer ports. When port selectors became available the switch was shelved, later to be resurrected as an *ad hoc* telephone switch, and subsequently to switch high voltages to control coin telephone functions. Its use as a telephone switch demonstrated the desirability of greater capacity for more interesting services. Serendipity provided a temporarily unused Redcom switch from another laboratory with which to experiment. Its greater capacity suggested providing services to other people. Its wealth of functionality, the capability to act as a central office, suggested greater service possibilities. Along with the acquisition of a true telephone switch came the desire to design truly innovative switching features and services, not just a better answering machine.

A crude system was put together in a matter of weeks, providing basic enhanced services. The immediate desire was to correct the problems experienced with currently available services, such as the lack of flexibility and feedback from call forwarding functions. As such improvements were put in place, new offerings were invented and the software was designed so that new features and services could be implemented quickly. The feature challenge was popular for a period of time, demonstrating how easily a new idea could be implemented, tried and perhaps discarded.

Obtaining desired services and interfaces from other telephone companies was not a speedy process measured by do-it-yourself standards. Answer supervision, an arrangement which reports back to the originating switch when the called party answers, took over 18 months to be installed. Mechanisms to report the originating number to the terminating switch will not be universal for years to come. Interim techniques were employed pending installation or propagation of such services.

Although these techniques provided less satisfactory results, they demonstrated the objective. For instance, when people were called without human intervention, a keypress was solicited in place of answer supervision. This allowed development of automated services to progress even under restricted conditions.

The initial system was somewhat unreliable. Service was frequently interrupted in order to install changes or demonstrate bugs. As more users came to depend on the telephone service, steps were taken to reduce outages. The first of these was the dynamic loading of structures that mapped telephone numbers to program names. This straightforward yet subtle tactic reduced downtime considerably because it obviated the need to recompile and reboot *bellerophon* each time a new service was introduced. Next, an independent program checked for the existence of the *bellerophon* process, reinvoking it if it was missing. This fell short of the desired goals, since it was not uncommon for *bellerophon* to hang. A more robust solution was implemented involving an audit primitive which caused *bellerophon* to issue a benign command to the MSP and receive a reply. The independent verifier issues this primitive and knows with relative reliability if the system is functioning. The next enhancement was to place a call from BerBell through the DDD network back to BerBell to verify the external interfaces. (One valuable lesson learned about the design of automatic auditors is not to sweep away evidence that is necessary for debugging. This mistake was made with the program described which killed the hung *bellerophon* process and invoked another. It was a mystery why the system occasionally rebooted itself, until the auditor was changed to produce a core dump of the hung program.)

Greater system reliability attracted more users. At first the mechanism for forwarding a subscriber's phone to a service involved the assignment of an additional number which terminated on the desired answering service program. This was costly in terms of dedicated telephone numbers. The solution, changing the forwarding algorithm to accept program names as well as telephone numbers, was a simple change which supported the notion of flexibility as well as eliminating the extra forwarding step and the dedicated numbers.

BerBell began as a monolithic system. Services were implemented within *bellerophon* using a myriad of special structures. The command interface originally supported a different command for each different service. The original idea was to generate interesting applications. The motivation to provide a clean and general interface came when other programmers wanted to write applications. The interface was defined empirically. A new application (call screening) was implemented with primitives that were defined as needed. Then one by one each application was rewritten using these primitives and defining new ones if necessary. It was rewarding to find that all but one of the primitives were defined by writing the call screening program. When all the services were changed to use the general interface and the old-style commands removed, the size of the command processing module was cut in half. Service application development proceeded independently of the switching program development. This started a trend of decentralization which was enhanced by the advent of *operator* primitives. New call processing algorithms are implemented by programs external to *bellerophon*.

8. Future Work

Decentralization will be pursued with the goal that all call processing be done by individual processes per port. *Bellerophon* will simply distribute messages received from the hardware to the appropriate processes and maintain a database to map extensions to processes. Processing modules will be distributed among processors communicating over networks.

There is interest in the artificial intelligence group to design parsers to process log files and create individual scenarios in English so that event paths can be easily identified and understood for the purpose of debugging or illustration. It will further be attempted to learn from these log scripts the events which lead to system errors and to predict and circumvent such conditions.

Another concrete goal is to provide administrative documentation and to package the system for distribution to interested parties.

9. Conclusion

The system described provides unconventional as well as conventional telephone service to a growing community of users. There is no limit in the potential for user programmable utilities. The next generation of consumers will require and demand to customise their products and services and they will have the educational background to do so. We expect that producers will abandon the current fad of featurism and provide value in terms of generality and well documented interface specifications. Service industries will provide end-user software as well as tools for do-it-yourselfers. BerBell has been, is, and will continue for the foreseeable future to be an extremely fun, captivating and rewarding project. To quote Louis Fyne, *Like the song says, it's a scientific lifestyle.*

Acknowledgements

Credit goes to all the users of BerBell. Their bug reports and design input helped mold it. I am particularly indebted to Peg Schafer who lived with it, Peter Langston who wrote a number of fun programs that use it, Don Ford who wrote *ttweather* and *statproc*, Adam Buchsbaum who wrote BERPS, and Stu Feldman for his moral and technical support and the patient editing of this document.

References

- [1] M. E. Lesk and C. A. McGonegal *User-Operated Directory Assistance* September 30, 1977
- [2] AT&T Marketing Sales Administration *An Introduction to the Bell System* 1975
- [3] Peter S. Langston (201) 644-2332 or *Eddie & Eddie on the Wire: An Experiment in Music Generation* USENIX Association Summer Conference Proceedings, Atlanta 1986 pages 19-27
- [4] Dave Hodgdon, Brian Redman, and Gordon Woods *Who Answers Your Phone When You're in the Information Age?* August 8, 1984

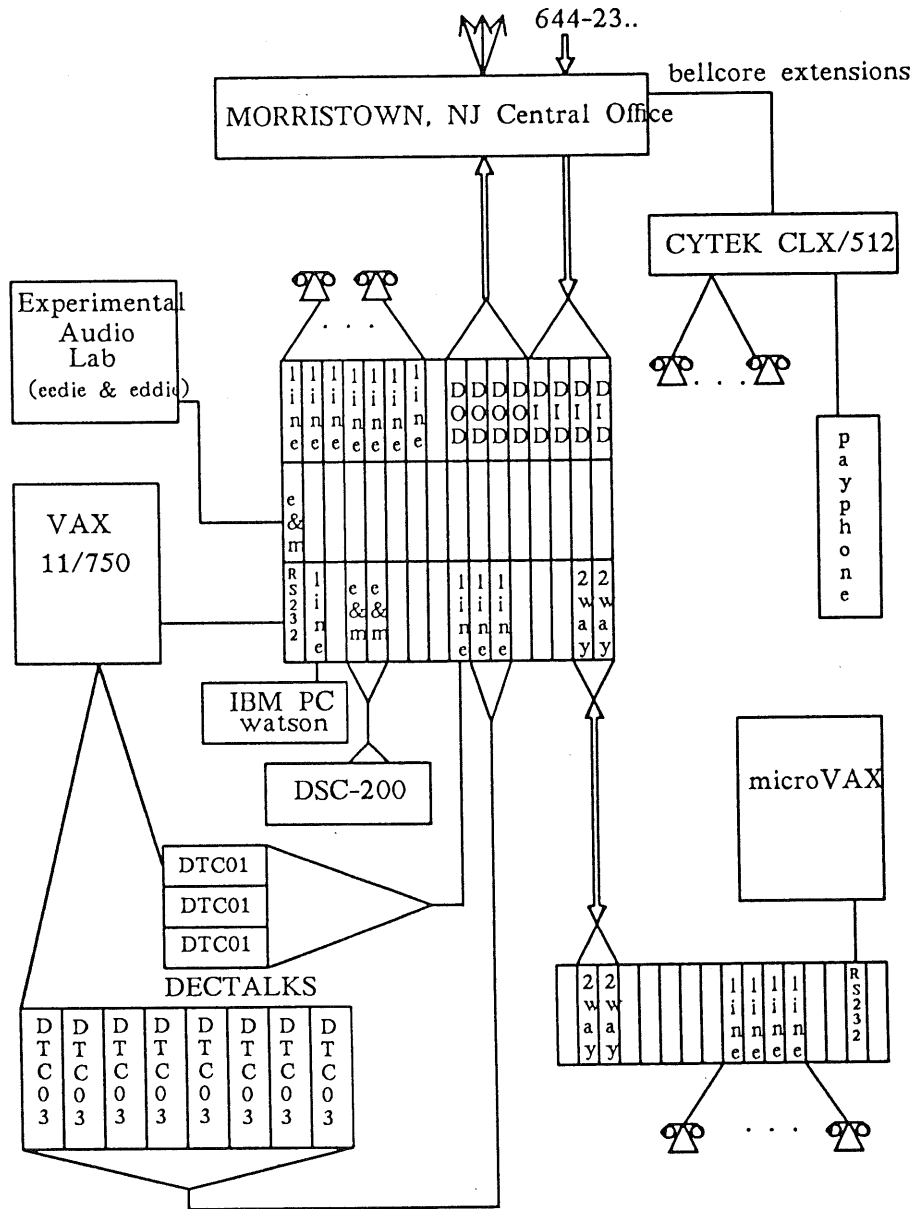


Figure 1. Schematic Diagram of Major Hardware Components.

A Day in the Life of Owles Hall

Helen Gibbons

*Business Manager
EUUG*



Helen Gibbons

It begins at 9 a.m. and it isn't easy!

To start with, some of our members living in the other European countries may forget about the time difference and telephone at 7 a.m. or even earlier. As the phones will probably have been switched through to my house for emergency reasons, I may well have had to take the call in bed! So if any of you *have* had the misfortune of asking me details of a conference while I am still asleep — I apologise now for anything I may have said!

The office functions officially from 9 a.m. to 5 p.m. GMT, with half an hour from 1 p.m. — 1.30 p.m. for lunch, five days a week. We don't work at week-ends, except at conferences.

Owles Hall is a large castellated building nestling in the heart of the Hertfordshire countryside. It is surrounded by fields of wheat, grass, rape (yes rape! which is actually a wonderful bright yellow colour in the Springtime) and barley, and fields of home bred deer, cows and horses. It ought to be very peaceful looking out on all that silent greenery, but somehow, the EUUGs manage to make sure that it is not. From the moment we walk through the door in the morning, phones are ringing, typewriters clattering, the printer deafens us with its churning out of email, people shout to each other seeking answers to obscure questions, the new franking machine (have you seen the little EUUG logo on our envelopes?) hammers its way through thousands of newsletter envelopes, the photocopier chunters out equally thousands of call for papers, someone is trying to hoover and collect the rubbish and the postman is hammering on the door with the latest batch of mail. We get about 50 letters a day.

Running the EUUG may seem very easy from the outside looking in, but believe me there is an awful lot to do. To start with we service five Executive meetings and at least two Governing Board meetings a year, plus meetings with UNIX Europe Limited, and Network Meetings which are held regularly. For all of these (especially the Governing Board) there is a great deal of organisation, and a small mountain of paperwork — the latest Governing Board minutes for example contained 32 pages and were circulated to 35 people — that is 1120 pages altogether. Now we also have a special Working Party week-end to organise in September and of course we have two major conferences a year to organise and run right from beginning to end and in every detail — they generate not only a mountain but a virtual *Everest* of paperwork. There is banking, invoicing, chasing bad debts,

keeping the books, making the VAT returns, visiting the groups, printing and sending out thousands of newsletters, generating membership, answering queries, and long long telephone conversations all over Europe. But none of it is grudged, we do very much like to be in personal contact with the membership.

Who are we, this dedicated team, forming the little central Secretariat around which the whole of the group made up from all the countries in Europe can function?

Well first of all there is me, Helen Gibbons, the Business Manager, and I suppose the day at Owles Hall begins when I walk into my upstairs office accompanied by my two great danes, Tarzan and Titan. They are guard dogs, keeping safe the EUUG records and eventually the new computer when we get it, and they lie under my desk while I work. I am however trying to train them to do something really useful for the EUUG, like answering the phone! But don't worry if you are thinking of visiting us — they have never yet eaten an EUUG member.



Jill Waite

Then there is the real backbone of the office, Jill Waite, my secretary. Her endless patience in dealing with all the back up typing, mailing, and organising is what keeps the office functioning and keeps us sane (or are we?)!



Bill Barrett

Bill Barrett most of you have already met at conferences. Bill works part time only, so doesn't tear his hair out quite at the same rate as the rest of us, (funny that because I think he has *less* hair than we have actually) but he does do a great deal of the conference booking work. He is now backed up on a full time basis by a newcomer to the office, Tina Wasyliw. Please be kind to her on the phone, she is just learning the ropes, but will be doing a great deal of the conference work for Dublin, and is operating our email. When asked what she thought of it all after her first week here she sighed very deeply, looked somewhat confused and said it was the busiest office she had ever been in!

Finally, there is Jenny Warren, who is a qualified accountant and deals with all the financial functions, including the endless financial reports required by the Executive Committee, and also the endlessly complicated transfer of payments from one currency to the other. Those of you who have had financial queries will probably have talked to her already.

And that really is how not only the day goes, but the weeks, months and now that we work six monthly from conference to conference, even the years. The day ends after everyone has gone and I walk down stairs, put out the lights, lock up and go home, silence reigns once again — and then just as I get to bed the phone rings — speakers from America, could I possibly give them the final details for the next conference, no they had not realised it was midnight — terribly sorry, but they had forgotten about the time change!



Tarzan and Titan learning to answer the 'phone

Apologia

*Michael J. C. Terry
mjct@inset.co.uk*

*The Instruction Set Ltd.
London, England*

Some readers of the EUUG Newsletter have pointed out that there was an inaccuracy in my article, "An Overview of the Native Language System", in Volume 7, No 2 of the Newsletter, 1987. In the article I wrongly stated that all X/OPEN group members' UNIX systems were obliged to conform to Issue 2 of the X/OPEN Portability Guide by the last quarter of 1987. In fact the X/OPEN members are obliged to conform to Issue 1 of the XPG by the last quarter of this year.

The implication of my statement was that all X/OPEN members' UNIX systems would support NLS by the end of this year, which is incorrect. What the statement should have said was that the X/OPEN members will eventually have to conform to Issue 2 of the XPG (which includes the NLS interface definition), but no date has yet been set for such conformance, and no date can currently be predicted, in that the members's UNIX systems are not yet even completely in line with Issue 1 of the XPG.

My sincerest apologies for this unintentional inaccuracy — I hope this recantation sets the record straight. So much for the enthusiasm engendered by a good idea like NLS.

While on the subject of inaccuracies, I also inadvertently got one of my troff string definitions wrong — consequently, the article refers throughout to the ANSI "XJ311" draft standard for the C programming language. This should of course read "X3J11". So much for dyslexia.

Otherwise, the article was spot on. So much for complacency.

The X/OPEN Native Language System Inside The Message Presentation

Pascal Beyls
xopen@echbull
mcvaxlinriatechbulllxopen



BULL

I joined BULL in 1979, and I am currently managing a UNIX *competence centre*. In addition I am Technical Manager inside X/OPEN representing BULL. I have been involved in Internationalisation since 1985.

Major event: On the 1st Jan 87, I got twin boys!

In the last EUUG Newsletter, Michael Terry from The Instruction Set presented an overview of the X/OPEN Native Language System (NLS).*

This paper relates the different thoughts that we had during the definition and the implementation of NLS. We concentrate on one of the major components of NLS which is the Message Presentation.

BULL, DEC and SIEMENS have jointly achieved the internationalisation of UNIX, as defined by the X/OPEN group.

The internationalisation of UNIX has been implemented by doing work in three distinct areas:

- allowing users to use new character sets. The ASCII character set is unacceptable in a language environment other than English, due to the number of accented characters and other symbols (the new ISO 8859 standard contains all the letters and symbols necessary used in Western European languages).
- allowing for differences in the different cultures (date formats and money symbols are two examples).
- allowing users to "talk" to the computer in their own language (the message presentation).

The Implementation

The X/OPEN group is committed to define only application interfaces. That means that a member is totally free concerning his own implementation. (In addition, as the administrative commands are not normally used by an application programmer,

* X/OPEN is a licenced trademark of the X/OPEN Group Members.

they are not defined in the Portability Guide). This point gives freedom and flexibility to design the implementation.

At the beginning of 1986, BULL and SIEMENS started an implementation of NLS from scratch. The main reason was that non-English people understand European problems better. Let's take one example:

French `spell(1)`: implementing a `spell` version for the English language is rather easy. But, even if you have a French dictionary and you have 8-bit cleaned-up the source code, your French `spell` would not work. Some reasons:

- All the verbs are declined: there are 5 large families of verbs with about 10 tenses and there are up to 122 exceptions.
- Large varieties of plurals for nouns and adjectives.

So, it was up to the Europeans to define and implement a European UNIX.

A first version of NLS was achieved by the end of 1986. At this time, DEC adopted and enhanced the Bull-Siemens implementation. A presentation of this implementation has been made during the EUUG Spring 1987 Conference.

Conformance

The X/OPEN members have committed themselves to deliver systems conforming to the interfaces described in the *Portability Guide*. The current commitment concerns the Portability Guide Issue 1 (by the end of 1987). This issue does not include the NLS interfaces. But it is the interest of each member to implement and deliver such interfaces as soon as possible.

The Message Presentation

The Message Presentation is a way to allow programs to interact with users in different languages. In the past, when a program was to be exported to a country with a language different than that used in the original program, the entire source program had to be re-read and all the messages translated into the new language. There are several disadvantages to this method:

- One has to have the program source in order to translate the messages.
- The new messages are hard coded into the program source. There must be one copy of the source for each language.
- Once the program is translated, the entire program has to be re-compiled.
- Each translated program becomes a new version of the program, and has to be maintained, which complicates the job of support personnel.
- Since the internationalisation has changed the source, you have to test the program to make sure the program logic has not been accidentally changed.

The Actors

Now, during the product life, we can consider three different people:

1. The *writer*: he writes the application in his native language. He ignores other languages, even he does not know if his application will be translated.
2. The *translator*: he has in charge the translation of the messages into another language. He doesn't know the application or its author.
3. The *user*: he uses the application in his native language.

The Requirements

The X/OPEN Portability Guide clearly states:

... the system must allow program messages (both input and output) to be handled in the native language of each user ...

There are several conditions that must be met in a serious solution for this requirement:

- The programmer must be able to program in his native language without having to worry about language problems.
- It should not change the way the programmer does his job.
- Translation of the program for different countries must be possible without using the program source. This allows you to have only *one* version of the program source, not one for every language. Errors added during translation of the source file are thus avoided.
- The same program on the same machine should be able to talk to several users in different languages at the same time.

The X/OPEN definition describes the way to choose the appropriate language by setting the environment variable LANG.

```
LANG=french export LANG
command
```

This is simple, very much in UNIX style. It is up to the user to choose his variable as he sets the TERM variable. We generally advise setting the LANG variable in a profile file. We can imagine to use the *comment* field in /etc/passwd to directly associate the user with his language.

Using the LANG variable

Does the message presentation allow a change of language during the execution of a command?

I would say no, because the usual case is that a user selects his native language (generally at login time) and does not change it later. We can imagine that a user may change his language within a session for some reason but I don't believe he would need to change his language during a process.

However X/OPEN allows such a mechanism. But in fact, the spirit was only:

- `catopen` returns a file descriptor to be used by `catgetmsg`.
- `catclose` has been added, only to be coherent with `catopen`. Only one `catopen` is used inside a program.

The example given by Michael (the user selects, within the same program, different languages) indicates this possibility. Although this example is attractive, changing the language at process time is not a major requirement.

Is the LANG variable enough?

Having set this LANG variable to French, the messages are presented in French language and if I use `spell(1)` to find my spelling errors on a document written in English, the spell would work according the French rules and not the English. So, a new variable (e.g. PROCLANG) should be necessary in order to define the language to be processed.

In fact, X/OPEN has not defined this variable. There is only an indication in Future Directions staying that PROCLANG is reserved in order to be used for this purpose. In addition, the ANSI C Committee has defined the function `setlocale()` which corresponds to `nl_init()`. With this function, a program may work on a file containing different languages (it is the case of an Arabic text which contains both

Latin and Arabic languages). So, there is no need to have this external variable `PROCLANG`. However, the announcement mechanism for defining different languages inside a file is not yet defined.

What is a message?

Sure, it is a stupid question and everyone will answer according the most famous message (`hello world`), but typically a message is a C character string, i.e. a string surrounded by quotes.

If a mechanism extracts automatically such strings in order to replace them by the appropriate subroutine calls, it would extract some *strange* messages.

For example:

```
# include <stdio.h>
FILE *fopen(), *fp;

main()
{
    fp = fopen("/users/example", "w");
    fprintf(fp,"%s0, "This is my message");
    fclose(fp);
}
```

An automatic extraction will give the following messages:

```
/users/example
w
%s
This is my message
```

instead of the actual message.

The selection of the *wanted* messages is made at compile time, which is not excellent. The best solution would be to differentiate the real messages from the strings inside the source code.

Mnemonics

During the implementation, we found the following problem:

Let's assume we have an existing program with an associated message catalogue containing translated messages.

What do we do to update the program without losing the translations already done, especially when some new messages are added between existent messages?

This problem of updating messages seems to be very important. We have to reuse the already translated messages and basically to "recognise" them. So mnemonics are needed. We consider it preferable to provide mnemonics and comments for the messages. A comment applies to a corresponding message. This feature would considerably help the translator. (See above the definition of the translator.) Generally, he will be far (both in time and in space) from the programmer.

So, comments and mnemonics are required inside the catalogue message.

The best way would be to directly include them inside the C program. It is up to the programmer to identify his message (by mnemonic and comment). Thus, we won't get any problems when updating a program.

How would this be possible?

1. Evolution of C language.

We can identify a message by simple reverse quote rather than double. Ordinary strings which are not to be translated (e.g. "file names", "r+"...) will be immediately dropped. For instance, the famous example becomes:

```
char *mess = `Hello world !` ;
```

I believe that this would be never accepted (changing the definition of C is a dream).

2. By comments inside the message.

— Inside the message itself using a syntax recognised by a translation tool. Now, the example becomes:

```
char *mess = "MNEM-comment: Hello world !" ;
```

— Outside, with a define.

3. Other mechanisms?

Are subroutines appropriate?

Every message in the source program is located and is replaced by a function call:

```
catgetmsg(catd, setnum, msg_num, buf, buflen)
```

or

```
catgets(catd, set_num, msg_num, s)
```

where *catd* is a file descriptor indicating the file where the messages are stored, *set_num* is the number of the associated set, and *msg_num* is the number of the associated message. This solution consists of replacing a char pointer with a call to a function that returns a pointer to the "translated" string. The messages associated with the programs are contained in a separate file. Tools can be made to help with the automatic extraction of message text and its replacement with call to the proper function(s).

This method has its drawbacks, however:

- Initialised static variables and global variables can not be replaced by calls to a function. These types of strings often represent 30 or 40% of the messages in a program.

```
char *foo = "This is my message0;
```

```
main()
{
    printf(foo);
}
```

Changing the definition of *foo* by a subroutine call does not work.

- Substitution of pointers by function calls engenders an overhead, namely an extra file descriptor and the time to read the messages from the file.

During 1986, Bull proposed and implemented another mechanism based on a new section in the COFF:

The mechanism of message presentation is integrated with the development tools: *cc(1)*, *as(1)*, and *ld(1)*. It is made up of:

- an evolution of some of their constituent parts, and
- a set of pre/post processors inserted into the development chain.

The mechanism of internationalisation is invoked as an option to `cc`. The programmer does not need to manipulate an intermediary work file.

The message presentation system has the following basic principles:

- A new section in the COFF is defined to hold the messages separate from program logic. All the messages in the program in the same language are grouped in the same section as defined in an extended COFF format. The message section(s) are included in the executable (`a.out`) file. The new section has type `message` and is identified by a new flag `STYP_NL` in the section header (see `a.out(4)`).
- An extension to the loader `exec(2)` loads into memory the message section associated with the user's declared language (environment variable `LANG`).
- There is a translation tool that helps the programmer (or a professional translator) associate the program's messages with messages in other languages, to facilitate the translation into multiple languages, without modifying the program source.

From the point of view of the programmer, this mechanism does not involve any programming interface. We could meet the requirements of the message presentation by such a mechanism. Also, the X/OPEN definition does not specify that message catalogues are files. In addition, it removes the different drawbacks inherent to a message catalogue built on files:

- The sending of a program (by `uucp`, for example) would also mean the sending of a message catalogue file for each language that the program should be able to speak. Without a message catalogue, the program is worthless.
- The program will only be usable when the message catalogue file is available. If it is located on a different mountable volume than the program, the program depends on two file systems, not one. A "cleanup" of the file system where the message catalogue is located effectively inhibits usage of the program, even though the program is still available.
- The message catalogue approach is not adapted for use with `.o` or `.a` files (libraries and archives). A separate operation is thus necessary during the link editing phase.
- Problems arise when trying to access the message catalogue. How to distinguish the message catalogue files for programs with the same name (`a.out`, for example....).

Conclusion

Given a specification, different ways of implementation are possible. That involves a very clear and well understood specifications.

The X/OPEN definition of NLS is, in fact, precise enough. Different implementations exist and prove the reality of these definitions. However, we have seen that although outlining the requirements of message presentation is rather easy, defining them completely is tricky. Although the NLS is a new definition, the X/OPEN group has made an excellent work by defining such interfaces in "*terra incognita*".

UNIX Standardisation: A Bystander's View

Brian Meek

mcvax@VAXB.CC.KCL.AC.UKIUF000

*Chairman, British Standards Institution Technical Committee IST/5 —
Application systems, environments and programming languages
King's College London (KQC)
Computer Centre (Strand campus)
Strand
London WC2R 2LS*

In one of my recent musings on standardisation topics, I made some such comment as "OSI has been the flavour of the standards month for more months than I care to compute". Suddenly, however, as when we realise that winter is turning to spring, or even summer, it seems that there may be a change of flavour on the way — from OSI to UNIX. (To get it all out of the way, OSI stands for Open Systems Interconnection and is not an anagrammed trademark of ISO which stands for International Organisation for Standards — and yes I know it looks as if it should be IOS — while UNIX is a trademark of AT&T Corporation.)

Operating systems have been an unstandardised mess for many years, with users forced to live with a particular supplier's product, or to add things of their own (which of course solves some problems but generates others). The need for a standard has been recognised for many years, and a few pioneers, with little encouragement or support from outside, have been endeavouring to develop an "OSCRL" (Operating System Command and Response Language).

Having behind it neither the glamour and politico-economic clout of OSI, nor the practicality of programming languages, this has remained a low-profile project and progress has been slow. Suppliers want to build OSI products, but they already have OS products. Users tend to think in terms of exchanging programs, not JCL — though micro users, wanting low-cost software to run on their low-cost hardware, have for some time shown that "runs under MS-DOS" or "runs under CP/M" is the sort of thing you might need to look for. However, even mainframe users should have realised that standards aid portability of people as well as programs. If we had an OSCRL in use for the last five years, how much would have been saved in retraining costs and improved efficiency? A great deal, I should imagine.

Given this background, and the development of 16-bit and 32-bit micros able to match the power of yesterday's minis, it is hardly surprising that people started looking to UNIX as the way to fill this aching void in the standards scene. It was reasonably portable, it was closely linked with a flexible and useful programming language, C, and despite being a proprietary product was readily available in various guises on a variety of middle-range machines which are widely used by a number of people. All the ingredients were there for a classic, bottom-up, product-driven standardisation project, both for the C language (now well under way) and UNIX itself.

Not being a UNIX user myself, I shall leave it to others to explain the interrelationships between the various UNIX-related standards projects — IEEE/POSIX,

X/OPEN, ANSI/ISO C, and the rest. I am more concerned here with wider issues like the relationship with OSCRL and the upper levels of OSI.

My concern about UNIX is not its standardisation *per se* — UNIX users obviously need it and should have it — but the idea of seeing it as the solution to the general need for a standardised operating system. Both the style of UNIX, and its intimate relationship with the C language, render it unsuitable for that role.

I have to stress that, so far as I am aware, the people involved in the POSIX project and the rest are not aiming at anything like that; they are after much more modest objectives, like being able to easily to transfer applications from one UNIX-based system to another, by providing a common standard for the interface. I am thinking rather of those not directly involved, who may pin higher hopes on UNIX standardisation than any of those concerned would wish to claim.

Many things are needed in this area, of which UNIX provides only part. As a user myself, and one concerned with providing services for other users, the way I see it is this.

We need a conceptual, generic set of definitions of the basic functions which any standard-conforming operating system must supply — not all the bells and whistles, but a set sufficient for 99% of the users for 99% of the time, and including all the functions needed to support OSI. (Perhaps these last could be left out for small stand-alone systems but I'm not convinced it would be worth the trouble.)

We need the standard OSCRL for use with all these functions, including a standard command for dropping down into the native command and response language, and a fully standard-conforming operating system would have to support this too (though see below).

We need functions and the OSCRL to come with parameters — user-controlled, not implementor-controlled — with standard default settings — one standard default being that you get the standard OSCRL when you log in. Function-related parameters would cover things like the usual file management variations such as whether old versions are automatically purged or automatically saved unless you say otherwise. Language-related parameters would cover things like whether you automatically get full responses or abbreviated responses.

Most important, we need a very clear concept of what the conformance of a product to one or more of these standards will mean, and a clear idea of how that conformance is going to be tested. Much of the point of having a standard is lost if you have to take adherence to it on trust.

Of course, users can also invoke operating system functions indirectly, through applications programs and utilities, as well as directly through a command language. Thus, in addition, a standard generic systems interface for these functions will be needed, and standard bindings of the interface to the relevant parts of standardised user-level applications tools (such as programming languages) which allow invocation of such functions.

With all that in place, you are then in a reasonable position to allow users to benefit from a coherent, standardised, but flexible systems environment. Note, however, that it is possible to provide the standard functions without providing the standard user interface (i.e. OSCRL), and even if the OSCRL is present you can get at the functions without it.

This is the context where I would see UNIX standards taking their proper place. So far I have talked about a standard basic operating system (or the framework for one). A standard UNIX-like operating system would provide all the standard

functions, but for conformance would provide them realised in a UNIX-like way. It would also provide any further functions which the UNIX community would expect to find, together of course with the necessary additional generic systems interfaces to allow users to invoke these indirectly (for example through a C program or library function, which is where IEEE/POSIX would come in). Finally, of course, it would have to provide a UNIX-style command and response language, such as many UNIX users are likely to be familiar with through the "shell". (A standard conforming UNIX-like operating system could of course support standard OSCRL as well, but it wouldn't have to.)

I can imagine some users of UNIX, who know and love only that, getting impatient at all this and saying "look, all I want is standard UNIX, I don't want to be bothered with anything else". I sympathise, but it has to be faced that UNIX does not live in an isolated world of its own, and that it entails some obligation to take into account relationships with other things.

The history of standardisation, certainly in our field, is littered with evidence of the problems caused by standards committees taking a narrow, introverted view. One of the great disadvantages of the product-driven style of standardisation is that for various reasons it makes it difficult to separate out levels of abstraction, to think of a concept apart from its actual realisation. A UNIX-like way of looking at and defining the systems functions to be performed by a standard OS might pre-empt the way that equivalents might be defined in some other, or a "generic", standard OS. That kind of matter needs to be looked at in a wider context; or, rather, at a higher level of abstraction.

The advantage of the approach I have outlined is that it allows development. It allows standardisation of one or more other, alternative, portable operating systems. (Pick one at random!) It allows additional user interfaces to OSCRL. For example, I am a "wordy" person (in every sense, I am told) so a keyboard-oriented OSCRL would be fine for me. A user "interface" involving peering at nasty little icons and fumbling with unruly mice is for me more of a user barrier. However, I fully accept that for some people it is, if you will pardon me, the cat's whiskers. I would not wish them to be deprived of the benefits of a standardised mouse-based diet, any more than I would take kindly to them imposing one on me. An interface has two sides: user-friendliness depends as much on the user as the system.

Now I readily admit that all the advantages do not lie with the "top down" approach I am advocating; it has dangers too. Too abstract an approach can lead to nasty collisions with reality when one gets down to the lower levels of the real world. This can lead to horrendous delays while you rethink the whole thing — and the top-down approach in any case holds an inherent danger of being lengthier than basing a standard on what already exists.

Cornelia Boldyreff of Surrey University has also pointed out to me, when discussing this sort of issue, that common underlying concepts might better be discovered by using a "bottom-up" approach, than to try to deduce them from abstract concepts which, if poorly chosen, might not even generate the needed functionality at all — you'll find your abstract model won't have a place for a useful facility. I wouldn't quarrel with that — and I accept that this can lead to *ad hoc* fixes which are likely to introduce irregularities and be a source of endless trouble in the future. She also agrees, however, that to proceed "by induction" from existing systems might fail to find something which happens not to be implemented — which echoes my earlier remark about pre-emption, especially if you start from only one style of existing system!

Cornelia has the advantage of being closely concerned both with language standardisation of C (she is convenor of the UK panel) as well as with POSIX, and makes the perfectly valid point that it would be as if no particular programming language could have been standardised before some generic language model had been standardised. I would agree there too — except that the world has moved on since the days of the first FORTRAN and COBOL standards. Then, concepts in languages that we now take for granted were still being researched and developed. High level languages were only a few years old. Operating systems, however, have been around long enough that, while development is still of course happening, there is a much firmer agreed conceptual base than would have been possible in the early 1960s.

However, I do not ask that UNIX should wait for a generic standard — only that the work should be done with an awareness of the general need, rather than in introverted isolation. The analogy with programming languages is sufficiently valid that the lessons of the dangers of introversion should be learned from from past experience in that area, as well as the more positive lessons from successes. There is no need for UNIX work to be delayed, provided it is consciously being done in the wider context I have outlined.

As well as (from our different standpoints) agreeing on that, Cornelia and I also agree that people should not make the mistake of trying to see UNIX as the whole answer to the standard operating system problem, or even the main answer. It is a part, maybe even an essential part, certainly a part which it seems can be usefully progressed at the present time — but no more than that.

Again I think that an analogy from programming languages is not too remote to be useful, and that the history of language standards is not wholly irrelevant. Imagine being told — today, ten years ago, twenty years ago — that “the answer” to the “language standard problem” was, say, APL. Or COBOL. Or Prolog. Or FORTRAN. Or Lisp. Or BASIC. Or ...

I rest my case.

(This article contains material originally drafted for an article to be published in Computer Weekly (Reed Business Publishing Ltd, Sutton, Surrey, UK) in a revised form.)

Demand Controlled Debug Logging



James A. Watson
...!mcvax!cernvax!paninfo!jw

Pansystem Informatics, Ltd.
Bahnhofstrasse 50
CH-8305 Dietlikon
Switzerland

Debugging and execution tracing by means of `printf` statements in programs is a time honoured, and much used, tradition among UNIX programmers. Indeed, many of the standard UNIX utilities include documented (or undocumented) command line options to enable various types of execution monitoring and debugging output. Sadly, the best known of these are the suite of `uucp` programs, which have as many as 10 different levels of debug output; unfortunately, there is no documentation of the differences between the various levels, nor of the meaning of the debugging output itself. That, however, is a topic that is best dealt with at another time ...

If one is to accept that using printed debug output is a permissible means of either debugging, or simply monitoring, the execution of a program, then it would be reasonable to try to make this activity as productive and reliable as possible. Some of the current approaches, and their problems, are:

- Add `printf` (or `fprintf`) statements to the source code during the development stage as necessary, and remove them as the various sections of the program begin to (apparently) work correctly.
 - Recompiling to enable or disable debugging is tedious, to say the least. If your program is fairly large, and your computer fairly slow, it is maddening.
 - Ending up with a program with no debug/monitor output is not always desirable. Programs that "appear" to be working often (always?) turn out not to be so, and the debugging statements often must be replaced.
- Bracket debugging output statements with `#if` (or `#ifdef`) `cpp` directives, so they can be selectively included or eliminated at compile time.
 - See previous statement about pain of recompilation.
 - See previous statement about "finished" programs not working.
 - Even if recompiling a finished program is not unduly slow or painful, it is quite conceivable that the target system for an application could have no C compiler, making it much more tedious to do.
- Make debugging output conditional on command line arguments, in the way `uucico` and friends are. This might be acceptable for simple monitoring of execution, especially for programs such as `uucico`, `uuxqt`, etc. which will generally run for a relatively short time each invocation. It has severe limitations for the debugging case, however.
 - In general it is difficult or impossible to predict that a program will fail at the time that it is invoked.

- When a program is running and has begun to fail, the act of stopping and restarting it (to enable debug output) often causes the failure to disappear (temporarily).
- Programs which run for extended periods of time may need to run for quite a while before the failure begins, thus they may produce a significant amount of output that is of no real interest.

The alternative to these methods is a system which allows debugging and monitoring output to be started on demand, repeatedly if necessary, without disturbing a running program. Such a system can be implemented using either named pipes (fifos) or sockets. Only the named pipe implementation is shown here, because the code required is significantly less.

Using the `O_NDELAY` flag, for non-blocking I/O, on named pipes, it is possible for a process to determine when opening a fifo for writing, or when writing on a fifo, if another process has the fifo open for reading. Thus, the output code can be enclosed in a test which checks to see if anyone is listening.

In fact, this turns out to be very simple, and requires very little code to implement. The following three routines, which should be saved in a file called `log.c`, contain everything necessary.

```
#include <signal.h>
#include <fcntl.h>
#include <string.h>

#define MAXNAMLEN 512

static int fd = -1;
static char fn[MAXNAMLEN+1];
static void (*sv)();

log_init (name)
char *name;
{
    if (strlen (name) > MAXNAMLEN)
        return (-1);
    (void) strcpy (fn, name);
    sv = signal (SIGPIPE, SIG_IGN);
    return (mknod (fn, 010644, 0));
}

void
log_write (buf)
char *buf;
{
    if (fd < 0 && (fd = open (fn, O_WRONLY|O_NDELAY, 0)) < 0)
        return;
    if (write (fd, buf, (unsigned) strlen (buf)) < 0) {
        (void) close (fd);
        fd = -1;
    }
}

void
```

```

log_end () {
    if (fd >= 0) {
        (void) close (fd);
        fd = -1;
    }
    (void) unlink (fn);
    (void) signal (SIGPIPE, sv);
}

```

This particular example is written for System V.3 UNIX. It would need a small amount of adjustment for other System V releases, and somewhat more for other versions of UNIX.

A program using this output facility would then look something like this:

```

main () {
    .
    .
    .
    log_init ("dbg_out");
    .
    .
    .
    log_write ("Made it this far...0");
    .
    .
    .
    log_end ();
}

```

When executed, the program would create a fifo in its present working directory called `dbg_out`. Subject to the limitations of the user's current `umask` value, this fifo would have read all read permissions on. As long as no one opens the fifo for reading, however, no output would actually be produced. Upon normal termination of the program, the fifo would be removed.

Opening the fifo does not require anything special; the `cat` program will serve nicely. So, execution with debugging output to the terminal could be done with the following commands (assuming the program is compiled to `a.out`):

```

$ a.out &
1537
$ ls .1 dbg_out
prw-rw-r--  1 jw      us          0 Jul 30 17:37 dbg_out
$ cat < a.out
Made it this far...
$

```

Obviously, the output of `cat` could be redirected or piped as desired, to save or print the logging information.

This method has been used on several major programming projects in the past year, with generally satisfactory results. It provides flexible logging of debugging or monitoring information, with relatively small requirements for program modifications. While there is some additional execution overhead, even when there is no output being produced, the effect is generally so small as to be unnoticeable.

Two significant deficiencies have been noted. First, there is no way to specify different levels of debugging output. This could be added, by making a read/write connection to the process, rather than a read only connection. The expense in complexity is quite significant, however. Second, in some cases it might be desirable to have several separate log outputs from a single process. Because the log file name and output file descriptor are held in simple static variables, this is not possible. Adding such a capability would not be extremely difficult, but again would produce a significant increase in the complexity of the package.

Book Review

Title UNIX System Programming
Authors: Keith Haviland and Ben Salama
Published by: Addison-Wesley, 1987,
 ISBN 0 201 12919 1.
Price: 15.95,
 Soft Back, 354 pp
Reviewed by James Malcolm
 University College,
 London

This book is about programming at the UNIX system call level. It is not about kernel hacking! I found it pleasant in style. Information is presented clearly and practically. There are lots of useful exercises and sample programs. Only one formatting error was found.

UNIX in this case means System V, but the authors take care to point out where the System V interface definition is different from other variants of UNIX, and also explain how some of the changes came about, so there is much that is generally applicable.

The reader will need a user level knowledge of UNIX, and will need to be quite fluent in C, so that constructs such as the following do not cause too many headaches:

```
#define SIG_IGN (int(*)())1
```

Because of this, I was somewhat doubtful about the value of the chapter on the standard I/O library, though I do learn some new feature of `printf` every time I read about it. There are no such doubts about the rest of the book. It covers operations on files and file systems, processes, inter process communication mechanisms, terminal handling, and screen manipulation (using `curses`).

I would recommend this book to anyone seeking to get to grips with low level programming in a UNIX environment.

UNIX Throws Up
or
How to spend two days on a boat and get nowhere



Alain D.D. Williams
addw@phcomp

Parliament Hill Computers
London

Alain Williams is an independent consultant specialising in UNIX and C. He enjoys visiting new places, meeting people, and being bought drinks; this is why he goes to as many conferences as possible.

Friday

After leaving through tobacco brown skies of Gatwick I arrived through bright blue skies with the sun sparkling off thousands of small lakes. The start of ten days away, out of reach of the 'phone — bliss. Pushing a trolley sponsored by Digital I passed an IBM stand; I haven't got out of the airport yet — give me a break!

Gratified to find that there were no customs to argue with over my excess wine, I swept out to meet Myriam and her friend Taria.

Myriam is an old friend whose job takes her to new and interesting places every year. I had been given a list of things to bring: olives and wine featured prominently. Trying not to be too over the top I had also brought a wine making kit; there are no restrictions if it isn't fermented.

The short trip past pine woods was filled with chat about mutual friends and my questions about Finland. I learnt that, officially, there are 60,000 lakes, the number had doubled recently as the definition of what is a lake has been changed. There are also 3000 offshore islands, I believe that that number doesn't change quite as rapidly. Helsinki is called the Daughter of the Baltic, has a population of 500,000, (Finland has 5,000,000). In 1812 it was made the capital of a Russian grand duchy, and gained independence in the confusion of 1917.

Enough history, the air smelt clean, I was with friends and I was going to have a good time.

Got home and celebrated with some of the wine that I brought. "It's not going to last long like this, better start on the home brew kit. The main thing missing is a demijohn, we'll have to buy one tomorrow."

Saturday

My pre-prepared excuse that Finnish time was 2 hours ahead of English time didn't work, no chance of an extra sleep.

"I want to see Helsinki. Be a grockle." We also needed some extra things for the wine making. On the way in to town I was informed that all foreigners living in Helsinki are either married to Finns or are spies.

One of the first things to get sorted out in a new country is whether the local ice-cream is worth eating. So we ate a, not so small, sample of Mövenpick also cloudberry with cream and lap cheese at the Academic Bookstore. Definitely edible. I wasn't too sure about the coffee though, it was *very*. Very *what* I wasn't quite sure, but quite definite. Something to beware.

On to Stockmann's, the largest department store in Helsinki. I could have been in London, Paris, or Rome, much of the same goods and brands were on display. After some searching we did find a very small home brew corner, it seemed to be aimed at quantity production.

On street corners *Kioski* abound. They vary in size from 5 to 15 foot square, but always the service is through a small hatch just large enough for a head and pair of hands. They serve sweets, newspapers, magazines, and cigarettes. They only just outnumber the ice-cream stalls. One interesting feature of the shopping area is that a large amount of it is built underground.

We dodged the trams and strolled down the Esplanade towards the old port, stopping off for a little something in the Kappelli, a large green summer house. The esplanade is a popular meeting place for the young of all ages, and is also frequented by the local lush who quaff vodka out of pop bottles. The high price of alcohol is an unsuccessful attempt to keep them dry. The curious thing was that most of them seemed to have a girl looking after them, the choice must be poor. I was informed that the Finnish girls call their men *Juntti* or roughly: yokel.

At the port the market was tourist board poster stuff, fruit (avocados cheap, cabbage not so), flowers, and reindeer skin rugs. I was intrigued by their idea of selling potatoes by the litre — "can I have the small ones please". The covered market provided us with reindeer kebabs (guaranteed Chernobyl free), to be washed down with strawberries (bought by weight).

More grockling: the Rock Church is a 1960's Lutheran creation on a small hill. The church is half buried and built out of a circle of red granite boulders capped with a copper roof. The Sibelius monument was next. This modern object is worse than the Chopin memorial in Warsaw. It looks like some organ pipes on legs, or perhaps a fistful of metal macaroni.

We went back home and had some Sima, this is a special May 1st drink made of fermented raisins, somewhat reminiscent of ginger ale.

"Let's go to the Dipoli and see who has arrived."

Nobody.

"Let's go to the conference hall."

This was hosting a Hungarian trade show, the hotel staff knew of it as a fashion show. The models walked out as I walked in.

We were presented with the usual bumph, and Ernő Rubik's latest game; he had just left — I was doing well. This game consisted of several clear plastic squares tied at the corners in such a way that it could be folded to different shapes and arrangements. It filled our spare time over several days.

The show was of everything from IC test systems to floor coverings, and frozen food to a PC clone running a Bible education program. There was even a System V machine from Videoton, they had a nice graphics CAD package, aimed at the house and factory building market.

Adjoining was a restaurant to which we returned for an advertised "Hungarian Evening". This largely consisted of a Brahms/Paganini style quartet (who took it upon themselves to terrorise a couple at the table next to us), and a Hungarian menu, the pancake and rum was good, the ice-cream flowed freely. We ate with Neil Todd and Bob Bishop. Neil was upset, having just been declared a non-person by the hotel staff: his booking had been lost.

We retired to the Dipoli bar and met the entire governing board. John Carolan finally persuaded the bar staff to serve us with some of the local vodka. (The Finns don't like serving. Waiters, bar staff, whatever, make one feel awkward just asking them for something.) The ladies became popular. It was noted that several people found the refreshments very much to their taste.

Sunday

We'd planned Sunday lunch at the Intercontinental as there you can eat as much as you want for less than £10. However that day was Mothers' day, one of the very important social dates in the Finnish calendar, so it was all booked up. Just to rub it in it started to rain. "First time for weeks."

That's what they say whenever I go anywhere.

So we drove off to look at the cobbled streets where they filmed Gorky Park. Many of the old buildings here date from 1900 and are of a large heavy Russian style, decorated in dark pastel reds, greens, and lots of tan brown. Many of the buildings have 3 sets of doors to protect against the cold winters; up until Easter one can drive over the ice to the small islands.

Monday

More rain, but now supplemented by lashings of fog.

Register. Nothing interesting at the Dipoli so bus back to town where, tiring of the drizzle and cold, we ended up in the Kappelli while listening to a live Jazz band next door. Here discussion ranged from finer points of C syntax and the shape of future conferences (Sunil Das had joined us), to the local beer and ladies.

Back from work Myriam and Tarya helped me to rescue Neil from the Russian hotel where he was staying. "What should I do with my copy of the proceedings?" We started on the week's important work: the wine making. We'd managed to find a plastic 5 litre container, that would have to do as a fermenting bottle. All the houses are hot: triple glazing and communal heating using cheap Russian gas, so no problem with finding a warm spot for it.

The house had a built in Sauna, relax, sleep well.

Tuesday

Down to the docks, and get on the boat. Boat is a bad word, too small, it looked like my local hospital afloat. It is huge, thirteen levels of deck. No time to stare — the conference was starting in the nightclub.

Jean Wood said hello, and Johan Helsingius used a jet-lagged Bill Joy as a dummy to show us how to don a life jacket, all explanations in Finnish. To prove that he had the nautical qualifications for this modelling job we were shown a film of him rescuing his car from the middle of a pond earlier this year. Bill recovered from that to tell us all what he thought would happen to work stations for the rest of this century. His basic message was that every few years you add a zero onto the end of every metric that you can think of, and that I'll soon have a Cray II sitting on my dining room table.

We recovered from that by having some sticky buns and tea. The Viking Line had done its homework well, all the paper napkins bore (in large red letters) the initials VI. Being an emacs user I disapproved. I decided that the coffee was as bad as at Manchester, but that the tea was all right once I had distinguished between the hot and cold water dispensers. Teus Hagen was the source of mysterious labels which read "Keep CALM batch", I never did understand what he meant.

We were then allowed into the conference room. Very swish, everyone had a "personal seat station" complete with table, hi-fi, and a buzzing button to annoy speakers with. To aid us in the latter task Johan's Penetron had provided us with paper dart material, thoughtfully marked with crease lines. These proved a boon in some of the later proceedings.

We started with a grep machine named after a Scottish mountain, and moved onto Rob Pike who wanted to abolish the concept of a line in UNIX. He claimed that lines were a hangover from the punched card, and that our thoughts were tied too strongly to that notion. Arbitrary records was what was interesting and he talked about a grep which handled that sort of thing.

Time to take the bags to the cabins. I was lucky enough to have one with a sea view. It was compact and had a small shower-room with a toilet that when flushed made a noise like a pistol.

After an explanation on how to read the ship's clocks (they all had two hour hands on them as we were to cross a time zone), Dominic Dunlop chatted about porting software between all manner of machines, amusingly illustrated with blood curdling slides from old films. The boat started its engines and almost shook him off stage.

Peter Langston spoke, as ever, to a packed auditorium. He rebutted the notion that entertainment is trite and a waste of time, it is a huge business. Entertainment provides many challenges to computing, and demands high standards: when did you last have an arcade game crash on you? "The point is that ..." Most people were just waiting for the demonstrations; which came. Eedie and Eddie, riffology, and Luxo Lamps.

After the talks, a little time to explore before supper. All the decks are named after birds: Penguin, Flamingo, ... The bar prices, while duty free, were by no means profit free. However, as much wine as you wanted with the meal was included in the price, more than one bottle was smuggled out to impromptu parties. I know some people prefer red, and some white, and some don't care as long as it is liquid, but the oddest justification ever for choice was given to me by Frank Kuiper: "My dentist doesn't allow red wine".

One of the nice things about the conference being on the boat was that nobody could stray too far, I kept on bumping into people I knew, some of whom I persuaded to buy me a drink. Gradually people vanished 'till just the faithful few remained in the disco. There was another conference on board, some Swedish social workers, who provided good company. It was said of one national group chairman: "You'll see the pass coming". To another we decided to award a prize in the disco trials for being so often on his knees to several pretty girls.

Exhausted, off to bed to dream of being chased by Luxo lamps dressed as football supporters.

Wednesday

I breakfasted with the sight of Sweden sedately sliding by, definitely novel. A contingent at the front of the ship claimed to have seen a Russian submarine dodging out of our way. Inspection of a few of those around me led me to invent

a new collective noun: a complaint of hangovers. Other damage from the night before was that the HP banner had been mysteriously inverted.

The other passengers began to clog up the lifts and strew the decks with suitcases; I was glad to escape to the warm dark of the conference room. Tanenbaum's talk* was very well attended. Minix was seen as something that brought a ray of hope to old kernel hackers who could not afford a System V source licence. I giggled at his justification for not implementing the nice system call: "A PC runs slowly enough, you don't exactly need it". The entire stock of *The Book* went from the Prentice Hall stand faster than copies of *Spycatcher* from New York airport.

The sun was shining as Jim Oldroyd and I stepped out to explore Stockholm. "I've spent weeks here recently, I'll show you around" said Jim. I boldly followed where I had never been before. We bought a cheap map, then found a bank where the displays told us that it had cost £3. After a circuit through some of the less exciting areas of town we found where we had been heading for. Out with the camera, play at being a tourist. After a quick rush past the Presidential Palace (or something) we ended up on familiar territory: a large city international style shopping centre.

Philip Dorn provided a much used target for the paper dart brigade. He succeeded in his goal of being provocative and held that the question was not so much whether UNIX had really grown up, but whether UNIX's acolytes had: more darts.

After supper we returned for the panel session. This consisted of Rob Pike, Brian Clark, Phil Dorn, Doug Michels, and Bill Joy. Nigel Martin (complete with dark sunglasses) played the Godfather and tried to maintain some semblance of control over his panel. The paper darts were now supplemented by wine bottle corks. Discussion ranged from "Did international standards organisations stop development" to "what is UNIX being used for". The only consensus reached was that the session should have been held in the bar...

which is where I found myself a bit later. The C++ zealots stuck it, I noticed their BOF still in full swing at 1 am (I am not sure by which of the two times held on board). Bill Joy found refuge with the space invaders machines. Others escaped to the disco, games tables, or bed.

Thursday

Being an enclosed sea the Baltic was largely calm. Being in a big boat any waves were small by comparison. This didn't stop Helen Gibbons complaining to me the following morning that she had spent the night in a sequence of boats being repeatedly shipwrecked. Over breakfast the boat slid into Helsinki.

The clocks had changed again, this time for the worse, one hour less in bed. Coffee cups proliferated amongst those who managed to hear it announced that Belgium and Iceland had affiliated to EUUG.

Stroustrup treated the C++ fans with some of his latest reflections: statements like "Human expectation is the only thing that grows faster than hardware".

I was beckoned out and asked two questions: "Do you mind making a fool of yourself, and can you sing?" I replied "no" to both and was shown the EUUG answer to the Eurovision song contest that had been broadcast a couple of days

* Reprinted page

previously. We had to clear the boat quickly at the end of the conference, this might help.

Teus thanked the local organisers, and presented Jean with a pair of water wings. We also said good-bye to Hendik Jahn Thomassen who was awarded the traditional EUUG Swiss army knife. Something went wrong: we sang and were asked to do an *encore*.

Johan had repeatedly been asked over the last few days what there was to see in town, he was adamant that "I don't know Helsinki — I just live here". In desperation he hired a bus and took us round, "The purpose of the trip is to prove that there is nothing to see here." It rained. He took us to Espoo to show us that there was nothing there either.

I bussed back to town with Rob, his life wouldn't be worth living if he didn't bring something home for the kids. How do people live there? A simple T shirt cost him £10. I later had a meal and met some people from the Netherlands. In an vigorous discussion we put the world to rights several times. We had the usual "enthusiastic" service, the waiter looked as if I had ordered the last of the dish that he was planning for his own supper.

Back home. The wine had started to ferment: bloub, bloub, bloub.

Friday

The birds outside my window were up bright and early, no doubt celebrating that the buds on the trees had burst open since my arrival. I was up not very bright, but what seemed early as I was booked onto the C++ tutorial.

Stroustrup was evangelical in his description of his creation. By the end of the day he had convinced me that I wanted to have a go, though I was a little worried about the portability of long externs that the translator seemed to generate.

A day's talking had not done his voice any good, so we sought throat soothing lotion at the hotel bar.

It was a busy place. People were leaving. Card swapping was the current game, all the people that one had *meant* to talk to but ...

Myriam and Tarya arrived, and we headed off to Katinka's in convoy. Being so near we felt that we had to try a real Russian restaurant. Full up. The Italian place next door let us in for an enjoyable meal. They served garlic, I fail to remember what went with it — but we all had some.

Saturday

A lazy day. Slope into town and again fail to find a demijohn[†]. I hadn't seen the surrounding countryside so we drove out past the *dachas* to a lake and woods. Once fuelled up at the inevitable ice-cream stall we ambled through the trees. They passed the biggest ant-hill that I have ever seen, I, however, got them all up my legs and needed to St Vitus's dance to get rid of them.

Sunday

One of the local tourist guides featured a picture of the Daedalus[‡] perpetual motion machine. I had wanted to see this for years so we went to the exhibition hall.

[†] Myriam bought the only one on a market stall a month later, but dropped it on the way home.

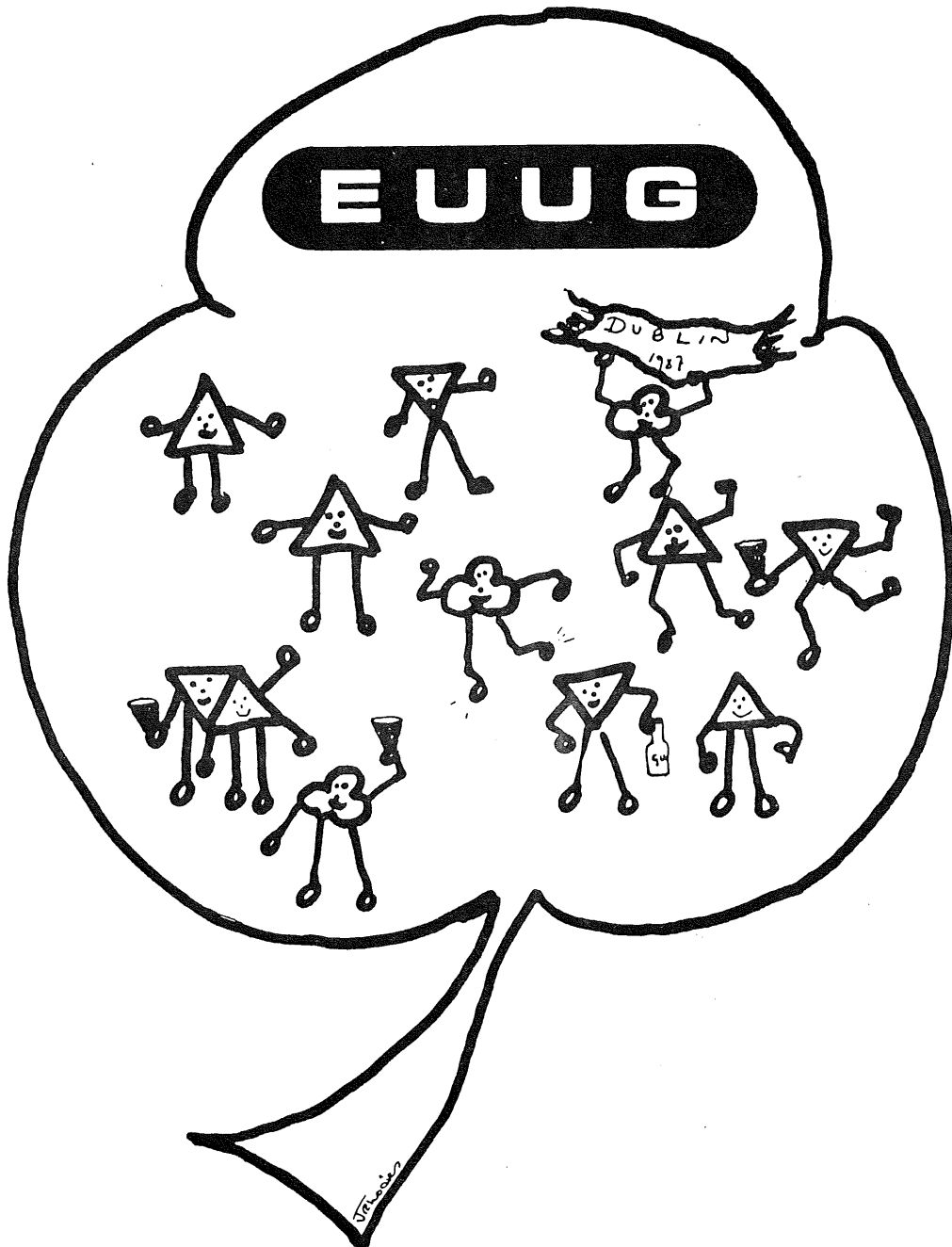
[‡] Daedalus is a pseudonym of a British academic, inventor, and columnist in a popular science journal.

The place was crowded.

There was also a separate fashion show, and a Russian trade show. The latter featured everything from Russian dolls to satellites. I discovered that many high tech items were made in cooperation with Finland or Japan. I asked a question about some machine tools and was followed around for ten minutes.

Myriam and Tarya dropped me off at the airport and picked up a friend just in from China. I wrote the last few postcards before dodging the DEC trollies, and seeing the last of this pretty country from the air.

I would like to thank all those involved in the conference organisation in helping me have an enjoyable week.



THE SONG FROM THE BOAT

The EUUG Ditty
— or —
The Pirates of Helsinki

This was sung as part of the finale to the conference. Those involved in the performance had, by then, lost all sense of dignity.

We arrived in Finland,
We didn't care
We went in search
Of a Polar Bear
Got in the boat,
Set off from the shore,
When the drink ran out
We called for more

CHORUS

Stick close to the source
And never go to C
And stay in tune
With the EUUG
(Repeat)

We went out fishing,
And caught a Pike
There wasn't much
It seemed to like
But LUXO lamps
They had a ball
So we brought them back
For a curtain call

CHORUS

Tried a 'phone call
To my friend
Got a DECTALK
In the end.
The panel session
Was such a thrill
Even more
From Rob and Bill

CHORUS

So, hackers all
Whoever you may be
If you want to rise

To the top of the tree
If you think that UNIX
Is a useful tool
Be careful to be guided
By our golden rule

CHORUS

For some strange reason this proved popular and an encore was demanded — for which the chorus seemed a little altered ...

Stick close to the source
And never go to C
And keep paying royalties
to A T and T.

With apologies to Messrs W. S. Gilbert and A. Sullivan.



Call for Papers

4th International Software Process Workshop Representing and Enacting the Software Process

Devon, England, 4 — 6 May 1988
(To be sponsored by ACM SigSoft and IEEE-TCSE)

ORGANIZING COMMITTEE

Gerhard Chroust	Mark Dowson	Watts Humphrey
Lee Osterweil	Dewayne Perry	Colin Tully

The 4th International Software Process Workshop will focus on executable or interpretable ("enactable") models of the software process, and their prescriptive application to directly controlling software project activities. A number of issues must be addressed if we are to develop comprehensive, robust models, together with environment architectures that allow their effective use. They include:

Process Structures

Generating useful prescriptive models requires a better understanding of actual software processes.

Representation Formalisms

Modelling requires model representation formalisms or languages with suitable syntax and semantics.

Limits to Mechanization

Formalization and automation of the software process should support and enhance human intelligence and creativity, not attempt to replace it.

Impact on Environments

Effective exploitation of prescriptive software process models will require suitable, model driven environments.

Prospective participants should submit a maximum 3 page position paper by 16 October 1987. Some participants will be asked to prepare short keynote presentations. Papers should be sent to:

LEON OSTERWEIL
University of Colorado
Department of Computer Science
Campus Box 430
Boulder CO 80309, USA
tel: 303 492 8787
e-mail: lee@boulder.colorado.csnet

or COLIN TULLY
STC Technology Limited
London Road
Harlow
Essex CM17 9NA, UK
tel: +44 279 29531
e-mail: tully@stl.stc.co.uk

Report from ICEUUG

Marius Olafsson
marius@askja.uucp

University of Iceland, Computing Center

The Icelandic UNIX Users Group (ICEUUG) was founded in November 1986. An enthusiastic crowd of some 60 people attended the first meeting, agreed on the laws and purposes of the group and elected a three member executive. Although many people attended the first and subsequent meetings, the enthusiasm waned a bit when time came to pay dues. The membership now stands at 10 (all Institutional members) but other membership categories are planned thus hopefully increasing the membership.

Meetings

ICEUUG has hosted three general meetings since November, on topics of interest to the Icelandic UNIX community. As most UNIX installations in Iceland are very recent, one meeting presented a basic tutorial on UUCP networking and the network in Iceland now consists of eight sites (of those there are four sites receiving Usenet) with more coming in the near future. Another issue that is of interest here, is the problems with internationalization of UNIX systems. One tutorial was given on how the X/OPEN group envisions the implementation of properly internationalised software. These meeting were well attended and more of similar tutorials on timely topics are planned.

In May, ICEUUG cosponsored a conference on UNIX in association with The Engineering Society of Iceland and the Icelandic Data-Processing Society. This conference was attended by more than a 100 people (good by our standards). Talks on topics ranging from the use of UNIX systems in integrated information processing systems in the Fishing Industry to the relevance of UNIX to toy-computers (or should I say personal-computers). Panel discussions were held afterwards which, as usual, degenerated into arguments as to whether UNIX is user-friendly or not (whatever that means).

Next Year

As the number of UNIX installations in Iceland is growing at a healthy rate (from 2 to 10 sites in one year), ICEUUG intends to increase its membership. Primary concern initially will be to promote the interconnection of the various UNIX sites for better flow of information between users of these systems. Even here in Iceland, there are communication problems and people have been known to spend many days on problems just to discover that someone else solved it across town.

ICEUUG is involved in the process of registering .IS as a domain on Internet in cooperation with SURIS, which is an umbrella organization of networking parties in Iceland and the hope is that this will be completed sometime next year.

Other areas that ICEUUG plans to involve itself in are for example the continuing problem with codesets and peripherals (would you believe that `[]{}|\@^` are all missing from the standard Icelandic terminal keyboard!) This problem has been brought to the attention of the standard bodies in the country and it is hoped that a new keyboard standard will be adopted that is more UNIX-friendly than the current one.

ICEUUG Executive

Gunnar Stefansson

Marine Research Institute, Reykjavik (gunnar@hafro.uucp)

Sigurdur Hjalmarsson

Hughonnun Inc, Reykjavik (shh@hugh.uucp)

Marius Olafsson

University of Iceland (marius@askja.uucp)

Title	C by Dissection
Authors:	Al Kelley and Ira Pohl
Published by:	Benjamin Cummings, 1987 ISBN 0 8053 36861 2
Price:	£ 17.95 Soft Back, 250 pp
Reviewed by	Andrew Elias University College London

I enjoyed reading this book, and enjoyed trying out some of the exercises. As an introductory book on C for the beginning programmer, it is friendlier than some, though the style is a little "academic". It is not, however, for complete beginners, but for those with a year or so's programming in some other language behind them. As a book for beginners it could have benefitted from the inclusion of flowcharts and diagrams, and possibly cartoons, for those (like me) who prefer to think pictorially.

The idea of teaching programs by dissecting example programs is not original. Most of the better books teaching BASIC or Pascal do just that. In anatomy, a major purpose of dissection is to reveal structures and their relationships to one another, and to provide experiences which help hold this knowledge in place and make it more "real". The dissections provided by the authors, consisting of text on a grey background, are not especially memorable or eye-catching, and can be very tedious to follow. The do-it-yourself aspect of dissection is provided by the exercises, but as they are left to the end of the chapters they are not well integrated with the rest of the text.

There is a lot of material in some of the early chapters, difficult for a novice to absorb "at one go". I would have preferred to have sacrificed some of the thoroughness in favour of more clarity. Some of the more advanced topics in the earlier chapters (scope rules, casts, storage classes, ...) could have been dealt with later in the book.

In conclusion, although I personally enjoyed this book, I feel there is scope for much improvement if it is to serve the needs of the truly novice programmer.

The BELGIAN UNIX SYSTEMS USERS GROUP:

a group is born!

Marc Nyssen
marc@minf.vub.uucp

Secretary of the BUUG
Vrije Universiteit Brussel



Marc Nyssen is Associate Professor at the Medical Informatics Dept., Vrije Universiteit Brussel, Belgium. Since 1978, he has been an enthusiastic UNIX user and as colleague of Erik Blockeel, the software specialist who introduced UNIX in Brussels in 1978, he works on biomedical applications. In 1986 he was a co-founder of the BUUG. In collaboration with AT Computing he teaches UNIX courses.

In Belgium, the need was felt to create an independent users group; in the course of 1986, some of us decided to give it a real try: the BUUG was founded, as a non-profit organisation. The constitution of the group was heavily inspired by the NLUUG-constitution (pity it had to be typed in all over).

The purpose of BUUG is to generate an independent meeting place for UNIX users, to create opportunities to learn about UNIX, and to meet people concerned with UNIX, in the image and in the spirit of EUUG and the other national groups.

In a few preliminary meetings, all this was settled and most important: tasks were distributed. The "Board of Directors" looks as follows:

Members of the board:

- | | |
|-----------------|-------------------------|
| President: | E. Milgrom, UCL |
| Vice President: | Ph. van Bastelaer, FNDP |
| Secretary: | M. Nyssen, VUB |
| Treasurer: | E. Blockeel, VUB |

Persons responsible for:

Network:	J.J. Quisquater, Philips M. Lacroix, Philips
Contact with industry:	S. Allemon, BIM
Publications:	J. Huens, KUL
Events:	J. Seldeslachts, CIG A. Wambecq, Bancontact B. Schroder, Diamant Boart P. Verbaten, KUL

The first meeting was held Monday 3 November 1986, at the V.U.B. campus, Jette. Prof. E. Milgrom presented the "Belgian UNIX systems Users Group" and explained its goals. Teus Hagen came over especially to represent the EUUG.

During the first year (1987) the following activities were planned:

- First annual General Assembly 6 February 1987.
- Setup of the Belgian branch of EUNET/USENET network.
- Integration with the EUUG.
- Publication of a newsletter (at least two in the first year).
- Holding a technical meeting in the Autumn.

There are two classes of members: *institutional* and *individual* members. For 1987 membership fees were fixed at the (low) price of 10000 BF for institutional and 1000 BF for individual members.

An information brochure was composed and the first newsletter was sent out in March.

The numbers of new members took off quite nicely, after a somewhat "slow start".

Belgians are rather conservative and they first consider quite attentively how they will benefit from new things (not only BUUG, also UNIX itself). At this moment, we have 80 individual and 35 institutional members, and this is increasing. On 6 February 1987 the first annual meeting, only accessible for members, was held.

On October 16th, a colloquium on "International Computer Networks in Belgium" will be held in Namur (see further).

BUUG got recognition from the EUUG and the membership fee for our startup year has been waived (we would be broke already otherwise).

To obtain more info, contact one of the board members, or the secretariat preferably by E-mail, at buug@vub.uucp or by traditional means at the address:

Marc Nyssen
BUUG Secretary
co MINF VUB
Laarbeeklaan 103
B-1090 Jette
BELGIUM

COLLOQUIUM ON
INTERNATIONAL COMPUTER NETWORKS
IN BELGIUM

Namur, October 16th 1987.

B.U.U.G.: The BELGIAN UNIX systems USERS GROUP

with the cooperation of EARN and RARE

and

with the support of FNRS-NFWO and the Belgian Ministry of Education

Aims

Within the scope of its technical activities, the Belgian UNIX Systems Users Group organises, in cooperation with the Belgian Earn Users Group and the Belgian ABUT-BVT-Rare Club (Rare = Reseaux Associes pour la Recherche Europeenne) a one-day colloquium is to be held on Friday, 16 October 1987, at the Facultes Universitaires Notre-Dame de la Paix, Namur.

This colloquium will be devoted to International Computer Networks in Belgium and, in particular, to the two main networks presently used in Belgium, namely EUNET (European UNIX NETWORK) and Earn (European Academic Research Network).

These networks are mainly devoted to the exchange of mail, of news and of files between computers in Belgium, with Europe and with the whole world. While Earn is exclusively oriented towards the academic world, EUNET interconnects both scientific and industrial organisations.

The main objective of the day is to give to the members of both scientific and industrial communities the occasion to meet, and to share information on principles as well as on practical aspects of these networks and, by these means, to improve knowledge and cooperation, and to promote the use of this sort of communication infrastructure.

The morning session will be devoted to general presentations on networks, on EUNET, on Earn and on the network evolution in the world. The afternoon session will begin with local area networks and UNIX; it will end with non commercial presentations of practical experiences on the development and use of networks.

Provisional Program

Morning Session

Introduction to Research Networks

Philippe van Bastelaer, FNDP, Namur

Services provided by Research Networks

Jean-Jacques Quisquater,

Philips Research Laboratories, Brussels

Presentation of the EUNET-Usenet Network

Marc Nyssen, VUB

Presentation of the EARN-BITNET Network

Fernand Benedet, ULg

World Situation of Networks and Evolution towards ISO-OSI

James Hutton, Rare Secretary General

Afternoon Session

UNIX and Local Area Networks

Elie Milgrom, UCL

Presentations of practical experiences on the development and use of networks

Open Discussion

Call For Papers

The second part of the afternoon will be devoted to short (20 minutes) presentations of practical and original experiences with or about networks. They could deal with implementation aspects (gateways, LANs, development of protocols, etc) as well as use of networks (such as significant achievements facilitated by the use of networks). Irrespective of the speaker, these presentations should not have a commercial character. The language should preferably be English.

Please feel free to submit such a presentation; they are essential for insuring a dynamic and participating aspect to the colloquium. In order to facilitate the job of the organising committee, interested persons are invited to send a one-page summary of their proposed talk before September 1, 1987; this summary should highlight the original and interesting character of the related experience.

Please, address your proposal to

Professor Philippe van Bastelaer,
Facultes Universitaires Notre-Dame de la Paix
Institut d'Informatique
21, rue Grandgagnage
B-5000 NAMUR

Registration

There is no charge for attending the talks. The price of lunch is 500 BF.

Copies of the proceedings will be sold at 500 BF (it will be free to BUUG members as it is included in their membership fee).

News from the Netherlands . . .

Frances Brazier
mcvaxlvu44lvupsylfrances

Patricia Otter
mcvaxlxirion!patricia

Affiliation Unknown

The latest NLUUG conference was held on June 11, 1987. This technical meeting was attended by 200 people, members as well as potential members.

With the theme "UNIX as stepping stone" seven technical presentations and twenty odd poster-sessions were held. The idea of poster-sessions was very well received. It provided a platform for informal exchange of views and ideas. Perhaps worth noting for the next EUUG conference !!

Abstracts of the technical presentations:

UNIX AND TEX — by Gertjan Vinkesteijn of Minihouse Holding N.V., Gouda

Knowing that the combination of UNIX and Tex has failed to succeed in the Netherlands, an overview of the author's personal experiences with the two, comparing the pro's and con's of Tex and troff will be presented. The extensiveness of the possibilities and of the macro's implemented within Tex are addressed, the final conclusions being that the quality of Tex exceeds troff.

NeWs - SUN's NEW WINDOW SYSTEM — by Rob Goedman of Sun Microsystems Nederland B.V., Soest

NeWs has unique properties:

1. it's expandable — it provides a basis for toolkits,
2. its "imaging model" (PostScript) is of a higher abstraction level than the usual grid-based systems.

These issues, along with comments on the design of NeWs, window management, the interaction with users and the client interface, will be addressed. The purpose of NeWs will be described and compared to similar systems.

X-STANDARDS — by Hewlett-Packard Nederland B.V., Amstelveen.

Abstract not received in time for the conference programme.

LIGHT MUSIC THANKS TO UNIX — by Floris van Maanen of the Sweelinck Conservatory Amsterdam.

After an explanation of the why and how, a demo will be given of software designed to follow Alex Manassen's rules of composition for the construction of two to twelve minute pieces of music. A pleasant (?) side-effect is that it can be heard on the designated equipment. The link to UNIX is both trivial and essential. UNIX has been used for debugging C-programs designed for the ATARI ST.

CAD and UNIX — by Jan Blaauboer of Intergraph B.V., Aalsmeer.

A short overview of the history of commercial CAD systems will be followed by an account of the advantages of using CAD on UNIX machines. A product overview was given.

REAL-TIME HOST TARGET EXTENDED DEBUGGING SYSTEM — by Piet Varkevisser of WestMount Technology B.V., Delft.

RHTX is a debug environment which can be used to debug programs in so-called "embedded computers". Often these are single-board systems which are incorporated in industrial applications. The RHTX debugger offers the possibility to debug various processes simultaneously in a real-time environment. Integration in the total software development environment is achieved by a uniform window oriented user interface, and by a careful definition of each of the individual tools and of the interaction between them. The host machine will be a SUN machine with UNIX 4.2BSD. The target machine may be any (board level) M68xxx system, providing a suitable support PROM can be inserted.

As could be expected, improvisation was required — a last minute cancellation by Amdahl was compensated by Apollo with a contribution on NFS.

Poster sessions included the following topics:

- ICE: an integrated programming environment for C, presented by students of the HIO "De Maere", Enschede
- Network Projects, presented by students of the HIO "De Maere", Enschede
- XLISP, presented by students of the HIO "De Maere", Enschede
- UNIX and Realtime, presented by G.C. Homburg, N.B.I. Integrated Solutions, Den Haag
- Cenix real-time operating system, using UNIX as a stepping stone, presented by Ben Dunselman, Nikhef-K, Amsterdam
- Controlling experiments in nuclear physics with UNIX, presented by Tom Ploegmakers, Nikhef-K, Amsterdam
- UNIX — Info, a new Dutch UNIX magazine, presented by Rene Akker, Sala Communications, Amsterdam
- The Nautus Accountings System, presented by R. ter Riet, SCB-HBO, Enschede
- MODPAS87 on the NCR Tower presented by R. Bats, SCB-HBO, Enschede
- C++ presented by A. H. Banen, SCOS Automation, Amsterdam
- Applix, an application generator presented by Ron Heusdens and Jossi Gil, Transmediair B.V., Utrecht
- UNIMS — Universal Information Management System, presented by Ron Vollebregt, UCOMS, Zwijndrecht
- Amsterdam Compiler Kit, an overview, presented by Michael Felt, Vrije Universiteit, Amsterdam
- Putting together a MacIntosh Application with ACK, presented by Michael Felt, Vrije Universiteit, Amsterdam
- MS-DOS Compatible via ACK presented by Michael Felt, Vrije Universiteit, Amsterdam
- UNIX and optical discs, presented by Erik van Ooyen, XTEC Computer Systems Nederland B.V., Waalre
- In addition to this all, the NLUUG had invited a publisher to provide a good selection of books of interest. This was greatly appreciated by the attendees: something to remember!

The NLUUG itself was present with a stand on which EUUG publications, X/OPEN guides and a number of other "important" documents were available for reference purposes. Needless to say all inquiries concerning the NLUUG, the EUUG and UNIX in general were answered accordingly.

Our next conference, on November 10, will include a vendor's exposition and parallel sessions at different levels. This conference will concentrate on UNIX, 4GL, Databases and Expert Systems. Speakers (from wherever they may come) are invited to contact the NLUUG: mcvox!nluug or +31-20-649411 (Patricia Otter).

Conference Announcement Planning Dates for the AFUU

Philip Peake
philip@axis.fr

Here are the dates of the next two meetings of the AFUU.

Convention UNIX '87
20th November 1978

This is a one day meeting to be held at the hotel SOFITEL in Paris.

The morning will be organised as follows:

Extrordinary AGM — Re-definition of constitution AGM (normal)

In the afternoon, there will be a technical conference:

Networking and Industrial topics

Convention UNIX '88
8th — 10th March 1988

This will be a three day conference, with an associated exhibition. To be held at the "ESPACE CHAMPERET" in Paris.

The conference topics have yet to be defined. Details of the exhibition can be obtained from:

B.I.R.P.
25 Rue d'Astorg
75008
Paris

Tel: (+33) (1) 47 42 20 21
Fax: (+33) (1) 47 42 75 68
Tlx: 643982 F

News from UKUUG

Sunil K. Das

*City University London
UKUUG Chairman*



Sunil K. Das was press-ganged into the Chairmanship of the UKUUG in 1984, and sentenced to further hard labour when re-elected in July 1987. His alter ego first encountered UNIX in 1977 whilst employed as a research fellow in the Computer Networks Research Group at University College, London. In 1980, Sunil joined the academic staff at City University's Computer Science Department, where his interests have included operating systems, local area networking and systems programming. His most recent involvement has been with a research project to develop an expert system using Prolog, which will help school teachers assess what learning difficulties are experienced by school children who have special educational needs, particularly with reference to reading skills.

The UKUUG held its Summer Technical Meeting over two days, 7th-8th July. The Programme Chair was Sunil K. Das from City University London, with Lindsay Marshall acting as our host at Newcastle University. Michael Lesk from Bell Communications Research was the International Speaker. Manufacturer demonstrations were provided by Steve Wanless and Ian Blagg of Sequent, and MARI (re: Newcastle Connection), while a brief talk by IBM was given about running BSD 4.x on the PC-RT.

A Business Meeting was convened during the Meeting to elect officers, present the UKUUG accounts and to report news from the EUUG. Sunil K. Das and Zdravko Podolski (the latter is now with Insignia Solutions Ltd. of High Wycombe) were re-elected as Chairman and Treasurer, respectively.

After a tour of the Computer Laboratory, the talks listed below were given over the two days. Where possible the author's abstract has been included with the title.

EUUG members interested in reprints of any papers should contact their National Group representative. UKUUG have sent a free copy of the proceedings to each National Group.

*Integrating the Apple Macintosh in a UNIX Environment**Nick Nei (Glasgow University)*

Work is underway in the Computing Science Department of the University of Glasgow to integrate clusters of Apple Macintoshes with clusters of UNIX machines. The Apple Macintoshes are connected to the AppleTalk local area network, and the UNIX machines (mostly flavours of 4.2 BSD) are connected to the Ethernet local area network. The challenge arises in integrating the alien PC environment of the Macintosh with a UNIX environment.

Many configurations are possible. A Macintosh can be used as a host on the Ethernet and thus behave (when connected) as an intelligent terminal emulator via its serial port. Files can be transferred between any hosts using ASCII file transfer or reliable methods like Kermit. More interestingly, a gateway between an AppleTalk network and an Ethernet network will allow any Macintosh to communicate using TCP/IP/UDP protocols with any UNIX machine on the Ethernet. A Macintosh can then connect to any UNIX host on the Ethernet using telnet, and transfer files using `tftp` and `ftp`. In addition, a UNIX host can be used as a file server to a cluster of Macintoshes, thus providing each with an enormous floppy diskette.

In the Department, an experiment is being carried out to use clusters of Macintoshes integrated into a UNIX environment for student teaching. This paper will report on some of the experiences and problems.

*Retrieval from Books and Maps: Lessons for Database Design**Michael Lesk (Bellcore)*

Every year sees more and more material in machine-readable form, much of it information conventionally available in other forms and used in other ways. Taking advantage of what the users know about this data offers great opportunities for improving the efficiency with which they use the result. Unfortunately, it also means that considerable specialization has to be put into the program. And thus generality conflicts with efficiency; and I know no solution.

To consider, for example, the results of the work on finding good driving directions in street maps, one could present the results at three levels:

1. Depth-first search is a better algorithm than breadth-first search for finding the shortest path in a graph that resembles a street map.
2. Watching the way people find routes is a better way to plan your program than reading mathematical papers that deal with a somewhat different problem.
3. You can't write a general purpose program to find shortest paths in any graph and expect to use it everywhere.

Perhaps the first of these conclusions is the most useful result, but the last is the most general.

In this paper, I will briefly describe three experiments done by finding large data bases and building interfaces to them. In each case, specific knowledge from the subject domain was needed to make a good interface, or to understand what was needed. More detailed explanations of the particular experiments are available in the literature.

*Experiences with MINIX and Networking**Jim Lyons (Newcastle University)*

With the arrival of MINIX — with its clean design — an opportunity arose to implement a network manager which would, like the file system and memory manager, be modular but retain the semantics of Version 7. As it was not the intention to re-invent the wheel, the decision was taken to base the module on an existing TCP/IP package — written by Phil Karn for the PC — and port it to MINIX using the V8 streams-based methodology. The complete implementation is intended as a vehicle for students to experiment with distributed systems within the Computing Laboratory of Newcastle University.

*I Come to Bury UNIX... and to Praise it**Dominic Dunlop (Sphinx Ltd)*

The UNIX operating system is being used increasingly as a vehicle for the delivery of application software. As such, it is usually almost completely buried: only when a user answers the login: prompt are they responding to a UNIX utility — all other interaction is with an application layered on top of UNIX.

But how deep and impervious should this layering be? What aspects of UNIX can form a useful part of an application while still avoiding the need to expose the user to such self-evident command strings as

```
lp -dletter -onb -oletenv -n2 ?
```

This paper, using examples taken from actual applications packages, explores a range of responses to this question.

*Opening Windows on UNIX**CSSD Newcastle University students 1986/1987*

This talk relates the experiences of a group of six postgraduate students who were given the task of looking at the numerous window management systems in the Computing Laboratory, with a view to improving the user's lot. The preliminary investigation took the form of a survey of all the WMS in the Computing Laboratory, a literature search and the evaluation of the potential of several systems upon which a WMS could be implemented.

The WMS in the Computing Laboratory were assessed against chosen criteria and from this many desirable properties of WMS were identified. The investigation led to the group prototyping some aspects of a visual interface to UNIX on X-Windows.

*Recoverable Object Management in Arjuna (using C++)**Graeme Dixon (Newcastle University)**Technology Forecast: Computer Science**Michael Lesk (Bellcore)*

What is happening in computer science? We see rapid progress in hardware, but less in software. The future should be many small machines, even more effort spent programming them, and even more frustration, unless we learn to do a better job

with distributed processors.

Presentation on the PC-RT

IBM

High Performance UNIX Multiprocessor Systems

Peter Lee (Newcastle University)

Recently, several symmetric multiprocessor UNIX systems have become available in the marketplace. Earlier generation multiprocessors were asymmetric, very expensive and offered very limited forms of parallelism. However, the latest generation of machines, characterised by machines such as Encore's Multimax and Sequent's Balance range, offer significant levels of true parallelism (20 — 30 separate CPUs) yet at a price which makes them highly attractive alternatives to conventional single-CPU UNIX systems. Thus it is believed that symmetric multiprocessors will become one of the dominant architectures in the very near future.

This talk will examine why multiprocessors are likely to attain this importance, and will describe the overall architecture of such systems and the means by which the parallelism they offer can be exploited. Some examples of the parallelism that has already been introduced into UNIX by the multiprocessor vendors will be used to illustrate the parallelism potential.

Improved Models of Natural Language for Consultative Computing

Eric Foxley and G M Gwei (Nottingham University)

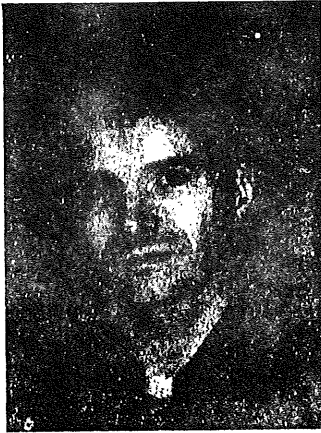
The restrictive form of language required in most computer systems is a handicap impeding the growth of interactive computing. A more language-mediated mode of interaction would alleviate some of the unnatural burden put on naïve users.

This paper explores models of natural language which attempt to extract more meaning from each interaction by exploiting the relationships between the various inflexions of a word, by extracting the separate parts of a compound (agglutinated) word, and by resolving the ambiguity of some words.

“EUUG... ‘European’, You Said ??...”

Michel Gien
mg@inria

Chorus Systemes



Michel Gien is currently vice-chairman of EUUG. He participated in the pioneering work on computer networks in the early 70's before getting involved in UNIX through a project to "rewrite" UNIX in Pascal... (what a funny French idea !!!) After 15 years in the research environment, at INRIA and then at CNET (2 national research institutions in computer science and telecommunications) he recently became one of the founders of Chorus Systèmes, a software company, developing a a new generation of UNIX distributed operating systems.

The theme of the last EUUG Conference in Helsinki was: "UNIX grows up ..."

As it is often the case in good conferences — and EUUG conferences are usually *very* good conferences — there was a panel session on the last day to give the audience the "word of wisdom" from the "gurus". The question put to the panel this time was — of course: "*Has UNIX grown up?...*" Of course, none of the gurus agreed with one another... and the audience had to build up their own opinion. So that was a good panel*.

May be no one is sure whether UNIX has grown up or not, but it is obvious to everyone — I think — that the EUUG *has* grown up...

The EUUG's 10th anniversary[†] is to be the main reason for the festivities during the next conference in Dublin this September. The EUUG is 10 years old, more than is necessary to be "à l'âge de raison"[‡].

Today there are more than 2000 members — 2300 to be precise, represented through 14 national groups. I have great pleasure in welcoming two new countries which have just joined the EUUG, namely Iceland and Belgium. Note that neither of these have lost any time in using the columns of this Newsletter to introduce themselves[□]. A group is being formed in Spain, the EUUG helping it get going.

* Participants can then report to their boss when they go back home: "You know, BJ said that UNIX is now a superconductor... And there was a big American guy, who also seemed to know what he was talking about, who said that UNIX still hates users... UNIX is surely the way to go, but may be we should still wait a little longer before we forget about MS/DOS..." And the boss goes: "Oh really? They said that? EUUG Conferences really help us to know what's going on..." And that's how you get a ticket to the next Conference.

† Looks like EUUG was born from Elvis' ashes ???...

‡ En Français dans le texte...

□ Could some of the other "older" groups take this as an example ...

From a "kernel" of individuals, the EUUG has grown into a large association comprising all European countries. Most of what are referred to as "EUUG members" are in fact organisations representing unknown numbers of people.

Aside from the quantity, the "quality" also has changed: from the bunch of university hackers, who created the group 10 years ago, the membership has diversified to now embrace all kinds of interests, including purely commercial ones. Some of the national groups which form the EUUG are becoming big and wealthy. They have their own interests, their own activities, and their own problems, they are mostly turned towards national needs.

The EUUG has had to adapt to reflect these changes. Clearly major evolutions towards much more "professionalism" in the services have been initiated. But, beyond the services, it is the overall purpose of the EUUG that, in some areas, may need to be revised.

At the last governing board meeting in Helsinki, these "feelings" crystallised into the idea of a workshop: a gathering together several responsible people from each national group, each representing a class of interests and/or services at the national level, who will "brainstorm" about the directions that the EUUG (and possibly the National Groups as well) should go. Conclusions should of course be reflected into EUUG (and National Groups) future services and policies.

This so-called EUUG "strategy" workshop will take place on the week-end of 5-6 September 1987, near Paris. It will start with the results of a survey of all national groups' activities, policies with respect to other groups, current objectives, ways of operating and directions. From that, discussions about what national groups should be expecting at a European level should provide a sound basis for an EUUG "strategy".

The EUUG has grown up... But it still needs to grow. The European idea is far from a day-to-day reality. European borders should open a bit further in 1992, and UNIX is contributing a great deal in overcoming borders. But lots of efforts are still required to get you, EUUG members, be totally aware that you belong to a European group and to get your National Group to actually contribute to Europe becoming a reality.

If you have any idea, suggestion, comment, or more generally input that you would like to bring to the EUUG "strategy" workshop, please do, preferably through your national representatives, or if you can't, directly to the EUUG secretariat.

... BON ANNIVERSAIRE ...

C++ Gossip

John Carolan*Glockenspiel Ltd.
Dublin*

John Carolan is the current chairperson of the Irish UUG. He is also managing director of Glockenspiel Ltd. of Dublin. Glockenspiel has been using C++ since 1985, and John has presented several technical papers on C++. His present work includes the development of C++ class libraries common between OS/2 and X-Windows on UNIX.

Bjarne Stroustrup easily won the competition for having the two most frequently bent ears at the "Boat" conference in Helsinki. Everyone who was interested in C++ took the opportunity to vent their favourite syntactic conundrums on Bjarne. Between mouthfuls of pickled herrings and weak beer, the perpetrator of C++ patiently answered every question, sometimes pausing to marshal extensive convincing arguments in support of why C++ should be the way it is.

Bjarne also gave a talk on how multiple inheritance, which rumours from AT&T suggest will appear during 1988, will look. However, it may be slightly different from the implementation suggested in Bjarne's paper. The paper, incidentally, was very refreshing in that it was one of the few papers at the conference which described genuine new UNIX research work which had been tested on real users (i.e. within Bell Labs).

Most EUUG and USENIX conferences now include sessions on C++. Bjarne's C++ tutorial at Helsinki was very well attended, so we are running C++ tutorials at the Dublin conference in September and the London conference next Spring.

USENIX are running a two-day workshop on C++ in November. It is being organised by Keith Gorlen, of the National Institute of Health in Maryland, who implemented a Smalltalk-style library in C++. This library, known as "OOPS" (short for object-oriented programming support), is available in the public domain.

(My opinion is that C++ does object-oriented programming in a C-ish way, while Smalltalk does it in a Smalltalk-ish way, and mixing the two results in unnecessary complication and overhead.)

If you want a copy of the OOPS library, you must send a 9-track tape to Keith:—

Keith Gorlen
Building 12A, Room 2017
National Institute of Health
Bethesda, MD 20892

If you would like to attend the USENIX C++ Workshop, mail Keith for details at

...!mcvax!usenix!nih-csl!keith

Preliminary info, *not yet confirmed*, is that the workshop will be in Santa Fe on the 9th and 10th of November '87. Closing date for abstracts is September 15th.

I know of at least five books on C++ under preparation, so EUUG members can give each other Christmas presents of C++ books which will, hopefully, be easier to read than Bjarne's. We will review each book in the newsletter as soon as it is published.

Two questions I have for readers of the newsletter...

1. Many universities are beginning to use C++ as a teaching language. I would like to assemble a list of people doing this in different countries so that EUUG can organise sessions and help swap information. Please mail me if you are teaching C++ at third level.
2. I don't know of any public domain C++ source other than Keith Gorlen's library. I hope to publish a list of contacts for source code or technical papers you would like people to know about.

C++ Tech Tip

If you have a class such as `Text` which is some kind of text object, you may want a `Text` to be usable wherever a string argument is expected; however, you may still not want the internals of `Text` to be accessible. If you provide a conversion operator, e.g.

```
char* operator char*();
```

things work fine when the `char*` is used as a constant, as in

```
Text foo;
....
puts( foo );
```

but something terrible usually happens if you do

```
Text foo;
....
gets( foo );
```

In this case, the conversion function is invoked and returns a pointer to some internal item in the `Text` object `foo`. `gets()` changes the string at that place, probably causing a core dump.

C++ grammar does not currently allow you to write

```
const char* operator const char*();
```

However, if you use a typedef, much sorrow can be avoided:

```
typedef const char* kstr;
struct Text
{
    ...
    kstr operator kstr();
    ...
};
```

Now, `puts(foo)` works fine, but `gets(foo)` gives you a syntax error instead of dumping at run-time.

Status Report on the Draft Proposed ANSI/ISO C Standard

*Cornelia Boldyreff
mcvax!reading!luoseev!corn*

*Gould Fellow in Software Engineering
Department of Electronic and Electrical Engineering
University of Surrey
Guildford Surrey GU2 5XH*

1. Recent Meetings

1.1 The ISO Working Group Meeting

The ISO/TC97/SC22/WG14 meeting was held during the same week as the X3J11 committee meeting in Paris, 8-12 June 1987, enabling international delegates to attend the latter meeting as observers. The WG14 meeting was the second meeting of working group; it was attended by delegates from Canada, France, the Netherlands, the UK and the USA; as well as members of the Japanese C Users Group and several members of X3J11 as observers. Following Steve Hersee's resignation, the new Convenor of WG14 is Dr P. J. (Bill) Plauger of the USA. It was agreed that the WG14 would continue with its goal of ISO/ANSI parallel progression of the standard. The ISO meeting consisted largely of reports on liaison activities and on the current status of the draft. Its major item of business was formulating a response to the Japanese comments.

It was agreed that WG14 would inform the SC22 Secretariat that it believed the appropriate forum for responding to these comments was X3J11. It was agreed that any changes as a result of these comments would be made to the current X3J11 draft rather than the ISO Working Draft (dated 1 October 1986) and that the ISO DP circulated would be based on the current X3J11 document.

Other national comments on the draft were discussed. The NNI (Nederlands Normalisatie Instituut) remain concerned about the definition of unary + in the current draft. They would prefer its removal. It was noted that disquiet was expressed at the BSI C Panel's Open Meeting in February 1987 on the definition of the preprocessor and the feasibility of its implementation. As WG convenor, Bill Plauger agreed to champion any papers from members of the ISO WG at X3J11 meetings in future.

1.2 Brief Report of X3J11 Committee Meeting

There were two main goals of the X3J11 committee meeting: to address international input to the draft; and continue responding to all public comments. A large part of the first two days of the meeting was given over to presentation and discussion of prepared papers including a paper by Bill Plauger addressing "Multi-byte Issues". In this paper, the object, letter, was proposed as a solution to the multi-byte character problem.

Plauger proposed the following:

A letter consists of 1 to MAX_LET characters (MAX_LET defined as 1 for ASCII) such that:

- A '\0' occurs only for 1 character, null;
- All C source characters are 1 character;
- A letter sequence begins with an initialised state.

TYPE `letter_t` is

- An integer type that doesn't widen;
- Has 0 for '\0';
- Can represent all letters uniquely;
- Can represent EOF uniquely.

Transform functions were also proposed to go from string to letter, and letter to string.

Plauger's paper was accepted as a solution in intent to the Multi-byte problem.

2. *Changes Made to the Draft in Paris*

Disclaimer

Below are noted changes to the draft made at the Paris meeting. This list omits many minor changes made which were editorial. These changes refer to the *unofficial* draft dated 15 May 1987.

2.1 *Type Based Aliasing*

(in 3.2.2.3)

All pointer references must refer to original object type (a type that differs from original object type only in its (un)signedness), or to a type that includes the original object type, or use a pointer to character.

Implementations are still able to do worse case aliasing if they don't carry around type information.

(end of 3.3)

All `lvaues` designating an object (whether or not the object is a member of an aggregate) shall have one of the following types:

The declared type of the object, a type that differs from the declared type of the object only in the presence or absence of the unsigned attribute, an aggregate type that includes one of the afore-mentioned types among its members (including, recursively, as a member of a sub-aggregate), or a character type.

2.2 *Hex Constants*

Valid hex constants now include both `0x` or `0X` for zero; and hex sequences of arbitrary length are now allowed.

2.3 *Name Information*

(3.5.5)

Name information holds across compilation units. When comparing types, two structures, unions or union types are the same if they have the same number of members, the same member names, the same member types, and for a structure, the same member order. For enumerations, they must have the same values.

2.4 *Pointer Comparison*

(3.5.2.1 and 3.3.6)

When two pointers are compared, the result depends on the relative location in the address space of the objects pointed to. If the objects pointed to are members of the same aggregate object, pointers to structure members declared later compare higher than pointers to members declared earlier in the structure; pointers to pointers to members compare equal to pointers to other members of the same union; and pointers to array elements with larger subscript values compare higher than pointers to elements of the same array with lower subscript values.

2.5 *setjmp*

`setjmp` must be a macro.

2.6 *Filename length*

A macro will be added to `stdio.h` which gives the maximum length of a filename; the POSIX name for this will be used.

2.7 *strxfrm*

(4.11.4.5)

It should return the length of string that it would have returned if it worked and returns in the first argument, the incomplete transformation.

2.8 *Enquiry on Composite Locale*

The user is now allowed to make an enquiry on a composite locale.

2.9 *Header Name Parsing*

(3.1.7)

The result of `/*` in a header name preprocessing token is undefined.

(3.8.2)

If the characters `\`, `'`, `"` or the character sequence `/*` occurs within a header name, the behaviour is undefined.

2.10 *Outstanding Issues*

Unary minus/Parentheses that group

Static/Extern Combinations

New keyword — `offsetof`

Macros with variable arguments

Multi-byte character/Letter additions

Currency information in locale additions

3. *Future Meetings and Projected Targets*

The next meeting of the X3J11 committee will be in Framingham, Massachusetts, USA on 14-18 September 1987. The current schedule of X3J11 is to complete responses to comments from the first public review. If this is accomplished and the draft standard can be updated in time, then it may be possible to go for a second public review in November 1987. This second public review will be for two months. It seems more likely that the revised draft will not be completed until after the December meeting of X3J11; in which case, the second public review will take place

in the first quarter of 1988.

The ISO Working Group on C has planned a separate meeting in Europe on 16-17 November 1987; the tentative venue for this meeting is Amsterdam. A major point of this meeting will be to ensure that international concerns are addressed in the ANSI draft; so the two standards, ANSI and ISO, continue to progress in parallel.

Whether or not there is an ANSI standard for C in 1988 will depend on the results of the second public review. If no substantive changes are made as a result of the second review, then the standard will be passed to X3 for processing; and ANSI approval could be achieved towards the end of the second quarter of 1988 or sometime in the third quarter. The end of the road is clearly in sight.

Book Review

Title	The UNIX System V Environment
Author:	Stephen R. Bourne
Published by:	Addison-Wesley, 1986
	ISBN 0 201 18484 2
Price:	£ 15.95
	Soft Back, 391 pp
Reviewed by	James Malcolm
	University College
	London

This book can be summed up very briefly: it is a revised edition of the author's "The UNIX System", which I guess most readers will know.

As before, it covers commands, editing (ed and vi), shell scripts, C, lex and yacc, system calls and libraries, [tn]roff, tbl, eqn, awk, ... in fact everything you might want to know as a user of UNIX. Many examples are given, including a complete system for maintaining the Bell Labs Murray Hill Tennis Ladder.

In fact the body of the text is very similar to the previous book: so much so that most of the index entries have page numbers only one different from before. The biggest change is that the page and a half on refer has gone.

Most of the text is generally applicable to UNIX variants, but the appendices (contrary to what is stated in the preface) definitely apply specifically to System V, and are much expanded over the original in consequence. Also, the appendix on the ms macro package is no longer there.

A large amount of information is covered in this book. This is both its strength and its weakness. It is ideal for a computer expert who wants to learn UNIX, and I still use it as a reference from time to time, but most beginners may prefer a more easy going approach.

POSIX Progress at ISO Level and BSI Level

*Cornelia Boldyreff
mcvaxlreadingluoseevlcorn
mcvaxlreadinglee.surrey.co.uklcorn*

*Gould Fellow in Software Engineering
Department of Electronic and Electrical Engineering
University of Surrey
Guildford Surrey GU2 5XH*

1. ISO Past and Present Action

In December 1986, ANSI proposed to ISO TC97, the ISO Technical Committee that deals with Information Processing Standards, a New Work Item (NWI) based on the IEEE Standard 1003.1 (Trial Use Standard for Portable Operating System for Computer Environments — POSIX). ANSI's aim was to facilitate international participation in this work leading to its adoption as an international standard. ANSI proposed that this NWI be assigned to SC22 — the subcommittee under TC97 on Applications Systems, Environments and Programming Languages.

The ISO ballot on this NWI did not close until the 20 February 1987. During this period several related items were under consideration within TC97. As a result, an ISO Special Working Group was formed to consider POSIX, System Software Interfaces and Related Issues (SWG/PSR). This SWG met in May 1987 after the POSIX ballot. The ISO Members were not unanimously in favour of the POSIX NWI; there were two main areas of concern: lack of a "language independent" specification of the POSIX interface; and concern over the IEEE trademarking of POSIX. The SWG/PSR passed several recommendations; two related to POSIX directly. It recommended that the POSIX NWI be accepted and assigned to SC22; a slight modification to the scope of the work item was made to ensure the POSIX standard provides a functional definition of the interface (it was recognised that initially this may be C Language flavoured and more abstract in a later version); and concerns over the IEEE trademarking of POSIX were resolved. The SWG/PSR also recommended there should be close liaison between the OSCRL and POSIX activities. (For details on OSCRL, see Brian Meek's article in this issue.)

The New Work Item on POSIX has received official approval by ISO/TC97 (July 1987); and it is expected that Member Bodies at the ISO/TC97/SC22 Plenary Meeting in September 1987 will give their support to the establishment of SC22/WG15-POSIX.

2. BSI and other National Standards Bodies

Once the ISO Working Group on POSIX is established, member bodies of ISO will establish corresponding groups at national level. In the UK, IST/5/15 — the BSI POSIX Panel — will be required to ensure that the UK contribution to this standard has an effective focus.

In practical terms, the POSIX Panel will provide expert input to the BSI Technical Committee, IST/5, advising on ISO ballots regarding POSIX and related issues; and Panel members will make a direct contribution to the work of ISO by participating in the progression of the POSIX Work Item with other international experts through membership of the ISO Working Group on POSIX (TC97/SC22/WG15). All members

of BSI panels act in a voluntary capacity usually supported by their employers. BSI panels receive no official support from the BSI.

I have undertaken to act as convenor of an "ad hoc" POSIX Panel in the UK prior to official establishment of the ISO Working Group on POSIX; and associated BSI POSIX Panel. The first meeting of the "ad hoc" POSIX Panel was held on the 4th of August 1987 at the BSI Conference Centre, Hampden House, London, Room G4 at 2:00 p.m.

Other national standards bodies in Europe which are participating Member Bodies of ISO will be in the process of forming their own equivalents of the BSI POSIX Panel; if you are interested in taking part in the work on the POSIX standard, you should contact your national standards body. In Belgium, IBN; in Denmark, DS; in France, AFNOR; in Germany, DIN; in Italy, UNI; in the Netherlands, NNI; in Switzerland, SNV...

3. *The Future*

The question regards POSIX that I have been asked most frequently is: "When will there be an ISO POSIX standard?". In their proposal to ISO, ANSI were required to list preparatory work with target dates; these were optimistic:

1/87	Working Draft
1/88	Draft Proposal
7/88	Draft International Standard

The current draft of the IEEE POSIX standard (P1003.1/Draft 11) is most likely to become the ISO Working Draft. The aim is progress the POSIX Trial-Use Standard to a Full-Use Standard and an ANSI standard within two years; in parallel with progress to an ISO standard. If all goes according to plan, then in two years time, there will an ISO POSIX standard. Of course, work on standards progresses iteratively; and it may take a bit longer for consensus on POSIX to be reached internationally. Certainly though there appears to be much enthusiasm for this new venture into the area of standardising of an operating system interface and environment for applications and a recognised need for a standard in this area; so there is some basis for optimism.

4. *Is POSIX The Answer?*

There has been much discussion within the standards community raised by the POSIX proposal regarding the desirability of defining a standard for a "generic" operating system interface and environment. Such a standard would provide the POSIX work with a reference model. In what follows, I have not summarised this discussion, but simply give my own views in the matter.

I think the crux of the issue is: there is a need to standardise the interface seen by applications running under UNIX-like operating systems; what is being standardised is a particular operating system interface. Here the work is no different from standardising a particular programming language. In both cases, the starting point is an existing entity; and in both cases, there may be various implementations realising the language or operating system interface.

The modest aim of POSIX standardisation is not to provide a conclusive answer to the "operating system standard" problem; it is simply to address portability of applications on a particular family of operating systems i.e. those based on UNIX. In this particular case, the interface is provided by external C data definitions including C library functions, and operating system calls.

It is as if standardisation of a particular programming language was held up until a generic standard of fundamental programming language concepts was defined at a higher level of abstraction than that realised in any concrete syntax and semantics.

It will not be easy to identify fundamental operating systems concepts; these are likely to be as elusive as fundamental programming language concepts. A "bottom-up" approach of standardising particular operating system functions is more likely to bring to light common underlying concepts; here by examining the particulars, common concepts may be abstracted.

Of course, there is the obvious difficulty that if there are no particular instances of a concept, it cannot be "discovered" by this inductive method. On the other hand, the "top-down" approach is equally weak, poorly chosen abstract concepts may preclude the deduction of a particular concept and its corresponding function.

I think the standards community would do well to gain experience from framing standards for particular operating systems interfaces before attempting to answer the "operating system standard" problem.

It seems more appropriate to allow both the work on defining higher level abstract generic operating system functions and the work on standardising what is in effect a family of UNIX-based operating systems to proceed in parallel.

POSIX Portability Workshop

October 22-23, 1987

Berkeley Marina Marriott

This USENIX workshop will bring together system and application implementors faced with the problems, "challenges", and other considerations that arise from attempting to make their products compliant with IEEE Standard 1003.

The first day of the workshop will consist of presentations of brief position papers describing experiences, dilemmas, and solutions. On the second day it is planned to form smaller focus groups to brainstorm additional solutions, dig deeper into specific areas, and attempt to forge common approaches to some of the dilemmas.

Jim McGinness
Digital Equipment Corporation
Continental Boulevard MK02-1/HIO
Merrimack, NH 03054

(603) 884-5703
decvax!jmcg
jmcg@decvax.DEC.COM

For registration or hotel information, contact:

Judith F. DesHarnais
USENIX Conference Coordinator
PO Box 385
Sunset Beach, CA 90742

(213) 592-3243
usenix!judy

EUUG National Groups

AUSTRIA (UUGA)
Dip-Ing Wolfgang Schwabl
TU Wien
Inst für Praktische Informatik
Gusshausstr 30/180
A-1040 WIEN
Austria

BELGIUM (BUUG)
Marc Nijssen
VUB
Medische Informatika
Laarbeeklaan 103
B-1090 BRUSSELS
Belgium

DENMARK (DKUUG)
Mogens Buhelt
Kabbelejvej 27B
DK-2700 BRØNSHØJ
Denmark

FINLAND (FUUG)
Johan Helsingius
Oy Penetron ab
Box 21
02171 ESPOO
Finland

FRANCE (AFUU)
Miss Ann Garnery
c/o SUPELEC
Plateau du Moulon
91190 GIF-SUR-YVETTE
France

GERMANY (GUUG)
Dieter Lengle
GUUG
Mozartstrasse 3
D-8000 MUNICH 2
Federal Republic of Germany

ICELAND (INUSUG)
Marius Olafsson
University Computer Centre
Hjardarhega 4
REYKJAVIK
Iceland

IRELAND (IUUG)
John Carolan
Glockenspiel Ltd.
19 Belvedere Place
DUBLIN 3
Ireland

ITALY (i2u)
Carlo Mortarino
Viale Monza 347
20126 MILANO
Italy

NETHERLANDS (NLUUG)
Patricia Otter
Xirion bv
World Trade Center
Strawinskylaan 1135
1077 XX
AMSTERDAM
The Netherlands

NORWAY (NUUG)
Secretariat NUUG
c/o Jan Brandt Jensen
Unisoft AS
Enebakkvn 154
N-0680 OSLO 6
Norway

SWEDEN (EUUG-S)
Hans Johansson
NCR Svenska AB
Box 4204
17104 SOLNA
Sweden

SWITZERLAND (UNIGS)
Prof. Wolfgang Fichtner
Institut für Integrated Systems
ETH Zentrum
CH-8092 ZURICH
Switzerland

UNITED KINGDOM (UKUUG)
Bill Barrett
UKUUG
Owles Hall
BUNTINGFORD
Herts SG9 9PL
United Kingdom



The Secretariat: European UNIX®systems User Group, Owles Hall, Buntingford,
Herts SG9 9PL, UK. Tel: Royston +44 (0) 763 73039 Facs: Royston +44 (0) 763 73255
Network address: euug@inset.uucp

UNIX Clinic

Nigel Horne
njh@root.co.uk

Root Technical Systems



Nigel Horne has worked solely on UNIX since graduating in 1980 from Westfield College, London (and to a certain amount as an undergraduate as well). He has been involved in UNIX from the early days of "real" UNIX, the days of `seek()`, `roff`, PDP11's (they didn't even have split I+D in those days), keys for typing in the bootstrap, through to today when there are System V, 4.3 BSD, industry standards, and just as much confusion as when it all started.

Nigel is now a Director of Root Technical Systems.

I've been asked by several people to discuss various systems administrators' tasks. In particular "how can I ensure the system starts my program running automatically when it boots?", and "how do I change a line from being a terminal into being a printer line?". The answers to these lie in the file `/etc/inittab`.

When the system starts up it reads this file and executes some commands based on what it finds in there. The system is said to be in one of 8 states at any one time: they are called run levels, and are numbered "0" to "6", and "s". When the system boots up, it will normally default to "s" (for single-user, or system) mode. This can be changed by altering the line which looks something like:

```
is:s:initdefault # Default Init State
```

However, normally this is not touched. All characters after the "#" are ignored, allowing the administrator to add comments. Ensure that you do this, otherwise a little tampering can cause a real rats' nest of trouble! After the system has come up (into this single user mode) you can alter the current run level to level "2" by typing:

```
/etc/telinit 2
```

provided you are at "super-user" status. There is an unwritten convention that level 2 is the multi-user mode. In most systems, "s" and "2" are the only levels used.

When changing a level the system reads the `/etc/inittab` file to find out what it needs to do when the run level changes. For example it needs to start login processes on each terminal. A typical entry covering this is

```
01:2:respawn:/etc/getty ttyp tt_9600 # Login process on Port A
```

The "01" is a unique identifier for that line, the 2 means "run this command when you are at level 2", the `respawn` says, "if that process stops running, restart it", in effect it runs another login process after you logout; and the `/etc/getty` part is the

program call, `ttya` and `tt_9600` being its arguments. The system does not wait for this command to terminate, otherwise only one person could login at a time! If you want this line to be temporarily disabled, change "respawn" to be "off", and the system ignores it. You could put more than one level in the second field, e.g. "23", to indicate that it is to be run at levels "2" and "3", but no other levels. If no levels are given, all levels "0" to "6" are included.

There are some special lines which are executed only at boot time, in order to allow the checking of file system consistency, mount file systems and so on,

```
rc::bootwait:/etc/rc 1>/dev/syscon 2>&1 # System Initialization
```

This runs the command file `/etc/rc`, sending output to the system's console, and waits for termination. If instead of `bootwait`, we used `boot`, then a similar situation to using `respawn` would occur, the system wouldn't wait for the `/etc/rc` to finish. As `/etc/rc` contains critical processes we must wait for it to terminate.

Two final points before giving a full-blooded example of an `/etc/inittab` file. When the system changes a level it kills off programs running at the old level, so for example

```
/etc/telinit s
```

terminates all the currently running programs, in effect logging everyone out and going single user, as the `/etc/getty` lines will not be valid at the "s" level. If the `/etc/gettys` are valid at levels "2" and "3", and we move from level "2" to level "3", users will not be logged out. Secondly, if you make a change to your `/etc/inittab` file, you need to tell the system that you have made this change. This is done by

```
/etc/telinit q
```

which orders the system to reread the `/etc/inittab` file.

Here is a cut down example `inittab` file from a machine running System V.2, with some networking enhancements. The system is set up such that levels 2 and 3 are multi-user modes, but the machine is accessible from the network only in level 3.

```
sy::sysinit:/etc/sysinitrc </dev/systty >/systty 2>&1
    # Run before entering single user mode
is:s:initdefault: # Default run level is 's'
rc::bootwait:/etc/rc 1>/dev/syscon 2>&1 # Run the /etc/rc script
sl::wait:(rm -f /dev/syscon;ln /dev/systty /dev/syscon;) 1>/dev/systty 2>&1
    # Tidy up
co:23:respawn:/etc/getty console tt_9600
    # The console line at 9600 baud
ta:23:respawn:/etc/getty ttya tt_9600
    # Port A at 9600 baud
n1:3:once:/etc/inetd
    # Network daemon - start this when going to level 3
```

You should remember that this is a cut-down version of the real thing.

Compare this with the `/etc/inittab` file on your machine.

The above applies to System V and its derivatives; this does not include BSD or Xenix.

If there's anything in it you don't understand, drop me a line courtesy of the EUUG office. That goes for anything else you'd like to know more about, any questions you may have, or any comments you want to make.

What's New With System V

*Andrew Rifkin
...Imcvaxluellapr*

*AT&T UNIX Europe Ltd.
International House
Ealing Broadway
London W5 5DB*



Andy Rifkin was born in 1959 in a small town outside of New York City called Brooklyn where to be a good "hacker" one needed to carry a large axe.

After graduating Cornell University he joined the UNIX Development Laboratory at AT&T Bell Labs in Murray Hill, New Jersey. Andy was very involved in the Release 3.0 development in particular the RFS product.

Currently, Andy is a Senior Software Consultant at AT&T UNIX Europe Ltd. in London.

This column will be a regular feature of the EUUG Newsletter in order to keep the European community informed of the latest AT&T activities and UNIX® System V advancements.

The primary goal of this first article is to create a starting point and familiarise the European community with AT&T's office in London, called *AT&T UNIX Europe Limited* (AT&T UEL), and explain AT&T UEL's relationship with the AT&T UNIX System Development Laboratory in Summit, New Jersey. It will also present an overview of the technology which AT&T has introduced over the past 18 months.

In 1984, in an effort to keep pace with the rapidly growing European UNIX system market, AT&T set up a local facility in London, England. This facility, called AT&T UNIX Europe Limited, was a strategic move in that the primary objective of AT&T UEL was to gain the acceptance of System V, encourage standardisation, and promote the development of UNIX System based applications. All in all, the efforts were successful, System V has emerged as the dominant UNIX System, UNIX System standardisation is now in vogue, and the UNIX System market has grown, now attracting the development of sophisticated applications.

Now that System V has been so well accepted, AT&T UEL is working to facilitate the growth of the System V community through marketing and licensing, and is one of several sources of training and consultancy. Presently AT&T UEL employs over thirty people, several of whom are from the AT&T UNIX System Development Laboratory in Summit, New Jersey.

* UNIX is a registered trademark of AT&T in the USA and other countries

By maintaining a close working relationship with the UNIX System developers in New Jersey, AT&T UEL is in the unique position of representing the latest AT&T System V technology. However, this is a two way relationship; AT&T UEL is very active in the European community through conference, trade show, and standard committee participation. This provides feedback to the New Jersey development organisations to ensure that existing and future products meet the needs of the European community.

A current problem which is impeding the growth of the UNIX System market in the European community is the lack of experienced UNIX System developers. AT&T UEL is attempting to ease the problem in two different ways. The first is by providing highly specialised training courses on topics including UNIX System Internals, STREAMS/TLI, and device drivers. The second is consultancy, providing help with difficult bugs, porting, and product management, in order to reduce product development time and help speed UNIX System products to the market place.

Turning to technology, UNIX System V Release 3.0 has fulfilled many of the market demands, especially in the networking area. Release 3.0 has introduced the STREAMS technology, which is serving as the backbone for current and future UNIX System networking products. In fact, several STREAMS based TCP/IP implementations are already on the market (e.g., Spider and Lachman). The fact that users can now purchase ready-to-use networking protocol modules will greatly ease and expedite the development of networking based applications.

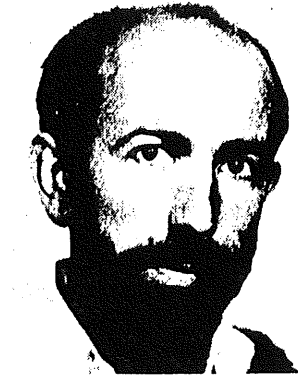
In addition to STREAMS, Release 3.0 contains the RFS distributed file system and the file system switch (FSS) architecture which allows the UNIX System to use a wide variety of file systems simultaneously. In all, Release 3.0 has served and is serving as the platform for future UNIX System developments.

An example of this is System V Release 3.1. This release uses the STREAMS technology to provide internationalisation features in addition to enhancing many of the features introduced in Release 3.0.

In July of this year AT&T UEL announced two additions to the Native Language Supplement (NLS) product family. These are the French and German Application Environments (FAE and GAE). These supplement the already available Japanese and Korean Application Environments. These products allow existing standard UNIX System applications to be used in non-English environments in a way that is compatible with both X/OPEN and ANSI-C standards.

Being an integral part of the AT&T European UNIX System strategy all AT&T UEL activities are focused on the growth and acceptance of UNIX System V.

The next issue will contain a more in-depth look at the System V technology in addition to an overview of the current UNIX System standards activities (X/OPEN and IEEE-POSIX) and their relationship to the System V Interface Definition (SVID).



EUnet

Peter Houlder
uknet@ukc.ac.uk

Computing Laboratory,
University of Kent

1. Introduction

I had hoped to include a lot of details about other EUnet networks in this article, but I have so far had limited success in extracting information mainly because the network administrators seem to be very busy people. Anyway Piet, Ruediger and Bjorn have given me a few details about the Dutch, German and Swedish networks, which are included below. The rest of this article covers some aspects of mail handling and routing, which is based on what we do at ukc and what Piet does at mcvox, but hopefully it has enough general points to make it interesting to all EUnet readers.

2. Some Information from the National Networks

THE NETHERLANDS

Piet Beertema (piet@cw.nl) is not only in charge of the Dutch Network, he is also the person who is in overall charge of the European Network (EUnet). The site mcvox has about 110 uucp links, about 50 of which get most of the news. The resulting network traffic, excluding local, campus and EARN/BITNET gateway traffic, now exceeds one Gigabyte each month. Besides the national and European links, mcvox now has links to the USA, Australia, Israel, Japan, Korea, New Zealand and Malaysia.

WEST GERMANY

Ruediger Volk (rv@unido.uucp) is head of the West German network, based at the University of Dortmund. There are around 100 active German sites and unido maintains a similar number of uucp links to German and international sites. About 25 German sites take news, so the overall throughput of news and mail exceeds 500 megabytes. Unido is also the "central node" (backbone) for the German EARN network.

SWEDEN

Bjorn Eriksen (ber@enea.uucp) wrote a recent EUUG article about the Swedish network. He explained his position to me as follows... "there is this guy, who heard about the uucp network, gets a connection to mcvox and soon after ends up as chairman for the national UNIX group", which seems to be the way things work, keenness quickly turning into responsibility. The site enea has 102 direct uucp connections, 123 registered sites and 31 news sites with 5 pending sites. It does not have direct links to other networks, although it is considering a direct connection to SUNET (the Swedish University network) to avoid intermediate gateway problems. The top domain for Sweden is .SE, which has 10+ subdomains, enea being the national backbone of the Swedish NIC registered domain .SE. The Swedish Defence Network also has a lot of UNIX machines, which are linked to enea via six sites of which f109a is the most prominent.

3. Mail Handling and Routing

3.1 General and ukc

People often ask me why their mail has got routed in an unexpected fashion. The simple answer is that we route on the information available in the world uucp maps, using the `Pathalias` program with some extra local post-processing.

The world uucp maps are maintained decentrally by area administrators in the USA and national backbone sites in the rest of the world. They are exchanged automatically.

As an example of routing, if we generate internal, or receive external, mail for a site "harry", then we look in a file called `paths` and we find an entry such as:

```
harry tom!dick!harry!%s
```

This says mail for "%s", standing for any user is to be directed to site "tom", who will forward it to "dick", who in turn will forward it to "harry".

Each night ukc runs the `Pathalias` program, which creates a interconnected graph of all the sites in the world maps. These are then scanned to work out the "quickest" route based on the lowest COST totals, as shown in the table below:

COST	VALUE	USE
HIGH	-5	(used to alter values below)
LOW	+5	(used to alter values below)
LOCAL	10	(local-area network connection)
DEDICATED	95	(high speed dedicated link)
ARPA	100	(used mainly by backbone sites)
DIRECT	200	(local call)
DEMAND	300	(normal call)
DIALED	300	(normal call)
HOURLY	500	(hourly poll)
EVENING	1800	(time restricted call)
default	4000	(if no cost included)
DAILY	5000	(daily poll)
WEEKLY	30000	(irregular poll)
DEAD	very high	(effectively non-existent poll useful for hiding links)

The following table should give an example of how routes are mathematically manipulated.

	COST			COST	
ukc	ARPA+HIGH	95	ukc	HOURLY*3	1500
tom	HOURLY	500	fred	DAILY	5000
dick	DAILY/2	2500			
harry					
		3100			6500

Ukc will, from the above, choose the `tom!dick!harry!%s` route rather than the `fred!harry!%s` route, because it is quicker even if, as happened recently, a site 16 miles from another site in the uk had its mail routed via `mcvax`, `seismo` and a Canadian site. That problem was corrected by changing the polling information, which bring me on to the subject of useful changes to your `Pathalias` output. If

a site polls your site, then they will usually expect to have their mail waiting for them on your machine, even if they may technically get it quicker if you route via another site. It is therefore desirable to remove explicit routing for sites directly connected to yourself. The use of the word COST in the above table is also unfortunate, as it is actually speed not cost related, so it is also a good idea, wherever possible, to route on free or at least cheaper channels. If small changes are made to polling information, one can often make large savings. In the above example a change from DAILY to DAILY/4 would cause routing via site "fred", which would save the ARPA (probably backbone) link, with its probable international charges. While we are discussing polling you may notice "default" in the above tables: this is based on:

Mail: tom, dick(DAILY), harry(HOURLY)

where tom would be given 4000. Do *not* use the form tom(), as this is a syntax error that will cause not only tom, but also dick and harry to be treated as DEAD.

At uucp we also route on to non-uucp sites, for example JANET, PSS and some other national backbones, but the rest of our international mail, for example US uucp sites, ARPA and BITNET sites goes through mcvox in The Netherlands.

3.2 Mcvox Routing

I am grateful to Piet Beertema for this information. The general method of uucp routing is as discussed above, but inter-network routing is somewhat different.

Routing to EARN/BITNET hosts is done using the routing tables supplied by the "central nodes" (EARN backbones) in each country. Routing to domain based addresses is done

- internationally on the top level domain, associated with a gateway machine.
- nationally the next-to-highest subdomain, which has a uucp or EARN name associated with it, for example cwi.nl is the domain based name for the uucp name "mcvox".

The top level domain for the Netherlands is .NL with currently some 10 subdomains, a number which is expanding rapidly. HEARN, the DUTCH EARN/BITNET backbone, acts for EARN/BITNET as the gateway into the .NL domain; it passes all unknown (unregistered or non-EARN) mail to "mcvox".

As uucp links are expensive mail from EARN/BITNET to a EUnet/UUCP site is routed via EARN to an EARN/BITNET-EUnet/UUCP gateway close(r) to the destination. For example mail from a French EARN node to a German UUCP node will be routed over EARN links (the same system is used to the USA with the psuvax1 gateway).

One extra small point which may be of interest: both "unido" (Germany) and "cernvax" (Switzerland) are both EUnet backbones and EARN central nodes.

3.3 Handling Other Network Mail

E-mail addresses are in general adapted to the network they are sent over; if necessary, "forwarding sites" (especially backbones) may change addresses to suit the needs of the network. In any case, Internet RFC822 addressing forms the basis for such changes (and for e.g. local representations). This implies that address forms that would result in illegal RFC822 addresses cannot be handled. Most notably this is the case for DECNET addressing, often used for local networks; such addresses have to be changed locally before they reach the "outside world": "foo::bar@host.uucp" is not a valid address, converting that to the form "bar%foo@host.uucp" may solve this problem.

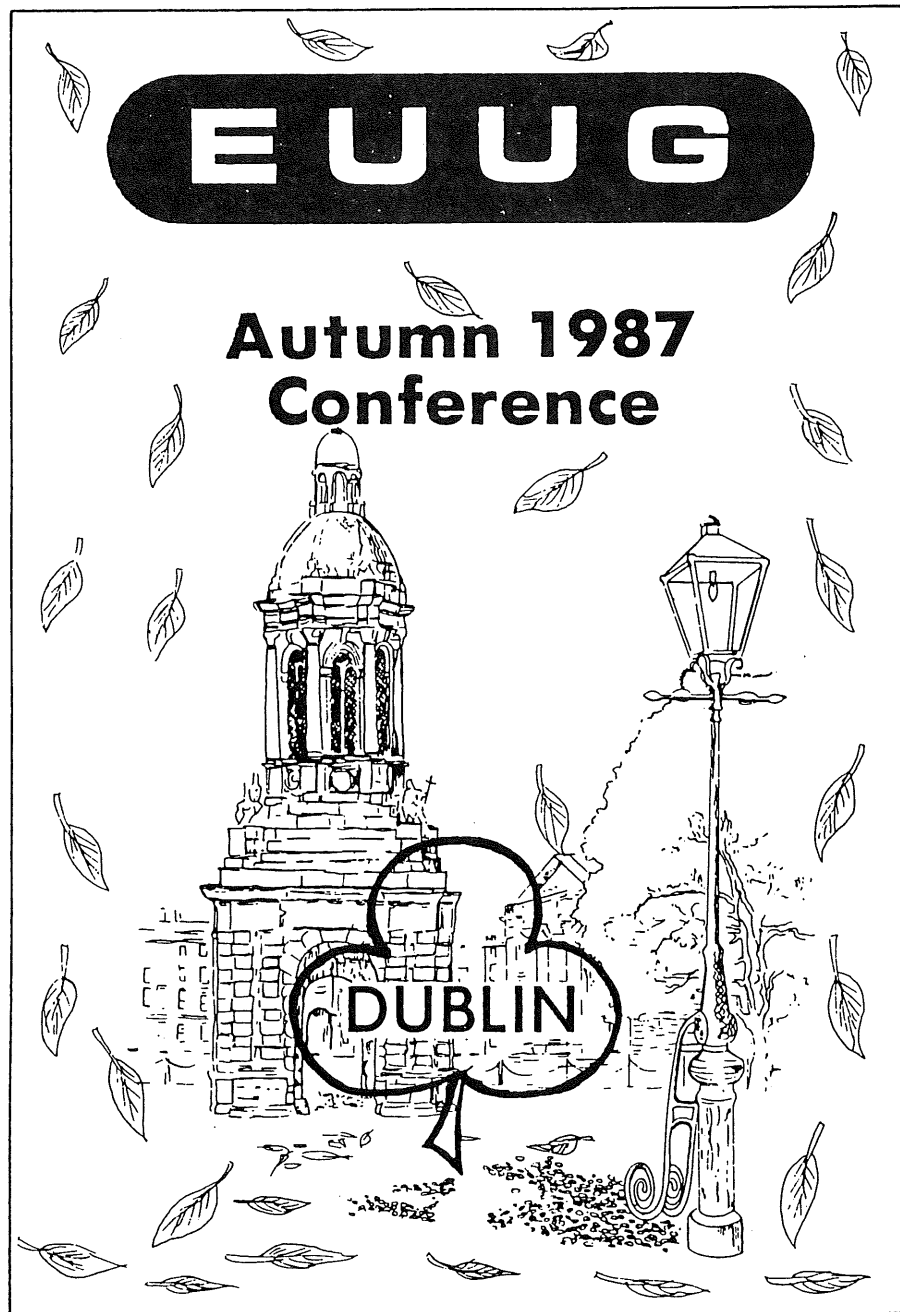
4. *Plea for Help*

I will shortly be contacting other national EUnet backbones for some more national information, but I would be more than grateful for any articles, paragraphs or even comments, which might be of EUnet interest. We would like this column to cover general aspects of networking and specific areas related to national networks, so please send any contributions to:

uknet@ukc.uucp

or

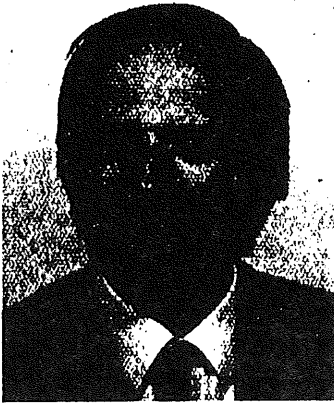
uknet@ukc.ac.uk



The X/OPEN Mid-Term Report

John Totman

Affiliation Unknown



John Totman is an electronics development engineer who has been involved in the engineering support, development and marketing of operating systems since the early 1970's.

He more recently strayed into UNIX when managing the development of commercial applications for departmental users.

A convert to the cause of standards and applications portability, John is currently a Marketing Manager for X/OPEN.

It is appropriate at this time to reflect on what X/OPEN has achieved in the first half of 1987, as we are already looking towards our 1988 goals as well as reviewing our plans for the remainder of the year.

How X/OPEN Operates

Like many companies, X/OPEN runs its operational business on a calendar year basis to achieve tactical objectives, with longer term strategic goals that set direction over the next 3 to 5 years. The strategic goals are defined by the X/OPEN Strategy Managers, a committee formed by a representative from each X/OPEN Group member company.

The Strategic Committee also sets and reviews the tactical objectives for the Marketing and Technical committees who define and manage work programmes that achieve the objectives.

It is these programmes of activity that create the "visible" aspects of X/OPEN, and 1987 is fast becoming the year of high visibility for the group both in Europe and the U.S.A. Our calendar of external activity closely follows that of the UNIX and standards arena, but with a few items of our own making added to further stimulate the marketplace.

I would view the two most significant developments for X/OPEN in the past six months as being our demonstration of application portability at Luxembourg in February, and our commitment to the development of POSIX as a full use industry standard.

Behind the scenes at Luxembourg

For the one hundred and twenty journalists, and the two hundred major users, software consultants, government officials, and system suppliers who came to the X/OPEN demonstration at Luxembourg, there is no doubt they saw the most professional and visually powerful presentation of applications porting that has ever been staged.

However, for those of us involved on behalf of the X/OPEN member companies, Luxembourg was much more an emotionally charged high point, seeing the results of two years of committee and project activity emerge with systems from all members on "stage together". For us, this was the most practical confirmation that X/OPEN could unite a fragmented group of companies to achieve a common industry goal. The resulting boost that Luxembourg has given us has enabled the group to move forward and tackle more and more ambitious projects, but more about that in the next newsletter!

POSIX — Why it is important to X/OPEN

Since X/OPEN publically declared its support for POSIX at UNIFORM in January, the X/OPEN Technical Committee has been operating a major programme of work to ensure forward compatibility and convergence of X/OPEN systems with POSIX. Already, major progress has been made through our work with IEEE on the early phases of POSIX.

For X/OPEN, the convergence with POSIX is vitally important, since it will bring a further one of the fundamental building blocks of our Common Applications Environment into the international standards arena. This can only be of immense benefit to computer systems users, since they will soon be able to reap the practical benefits of a complete working set of international standards by specifying X/OPEN as a single procurement requirement. In other words, X/OPEN would become a shorthand way for users to say "please supply me with a system that supports my application with a comprehensive and integrated set of functions and facilities that conform to internationally accepted industry standards!"

Clearly, X/OPEN still has a lot to do to arrive at this point, but by following its programmatic policy on standards, (which is to *adopt* existing standards, or *adapt* emerging or de facto standards or ultimately *develop* new standards where appropriate), X/OPEN can expect to make rapid progress in achieving this goal.

Book Review

Title: troff typesetting for UNIX Systems
 Authors: Sandra L. Emerson, Karen Paulsell
 Published by: Prentice Hall, 1987,
 ISBN 0-13-930959-4

Reviewed by Jaap Akkerhuis
 ...mcvax!jaap
 C.W.I.
 Kruislaan 413
 Amsterdam

Goal of the book

The book is aimed to be an introduction to the use of `troff` to the novice and also to be a reference manual for experienced users. It tries to correct the lack of adequate end-user documentation for `troff`. Alas, any explanation about the concepts of `troff` — or any other formatting programs is missing. For instance, the term "partial collected lines" is used a lot but never explained.

For the introduction to `troff` the authors explain all the basic requests and how to write macros. It is a pity that they do it in a haphazard way. Often they use a request, like `.de`, with the remark that the full details will be explained further on in the book. This is sometimes rather confusing. Apparently the authors did not have a clear idea on how to introduce a novice to the game of `troff`.

What I do like is that they give a full treatment of the `.nx-`, and `.rd-request`. Hardly any of the existing literature explains the possibilities of creating form letters with `n/troff` using these requests. Also, every possible `troff` request is explained, each description accompanied with an example of its use. But for the more experienced user there is not a lot new. Even small tricks, for example, what you can do with the `.ss-request` are not explained. Fancy techniques, like how to do balanced columns, are not handled at all. The chapters about the preprocessors and macro packages are sketchy and don't give more information than the existing literature.

To be a reference manual, it should at least replace the original `n/troff` reference manual. Some finer points haven't been covered, like the full definition of certain requests, for instance, the append to macro command: `.am xx yy`. So, don't throw the original manual from Ossana out of the window, you will still need it.

Typesetting

The most disturbing and misleading thing about the book is the title. Apart from a remark like "You should think as a typesetter", there is nothing in the book about typesetting or the noble art of typography. All the examples deal with the standard non-interesting cases of typesetting.

The typesetting of the book is not really done exceptionally well, it is just another book which is typeset by the authors. I'm always wondering why authors don't ask advice from a typographical consultant, it will do miracles for book design. Of course, this is partly the failure of the publisher. These firms are more and more interested in making money by cranking out printed paper and not caring at all

about how the product looks. I'm afraid that ignoring the issues involved with typography in this book will lead to even more horrible looking books than there are already around.

Errors in the book

In general there will always be errors in books. In this case, the advanced troff user will spot them easily, but for the novice they must sometimes be very disturbing.

The first one pops up in the first example in the first chapter (Page 3 & 4). This one can be waved away if you consider that novice shouldn't be hampered too much with details, but the next example (Page 5) is unforgivable. The quoted troff source of .PP for the *ms* macro package is missing some back slashes! This demonstrates again that it is not always easy to write about a tool by using it. There are more places in the book where these things happen. When showing the pitfalls of the arithmetic in troff using the .ll- request the complete promised test file isn't around. Some parts of how the file might have looked and some of the (incorrect) output is shown. Something really went wrong there.

Who should buy the book

Although I'm not very impressed by the book, it can be of some use for a lot of people. There are a lot of UNIX systems around which don't provide the original documentation. For these cases, the book fills a gap. Also, people complaining about the terseness of the original reference manual might want to read it.

Book Review

Title: Document Formatting and Typesetting on the UNIX System
Author: Narain Gehani
Published by: Silicon Press, 1986
ISBN 0-9615336-0-9
Hard back, 364 pp

Reviewed by Sally Rutter
sjr@inset.co.uk
The Instruction Set Ltd.
London

Preparing documents under UNIX falls into three areas: editing, formatting and viewing. Gehani discusses formatting, using `troff`, in detail. He gives a good description of the preprocessors `pic`, `tbl` and `eqn`, and of the `mm` macro package and the reasons for its use. `troff` is discussed only to the degree necessary to remedy the deficiencies in the `mm` macro package: as befits a book on typesetting, `nroff` is hardly covered at all.

Although the book is aimed at the novice user, the depth of coverage would be of use to an experienced user.

As Gehani works at AT&T, this book could almost be considered to be a user-friendly AT&T manual. Certainly, the very latest versions of the preprocessors are described. This can be annoying to someone with an older version. The worst chapter for this is the one on `pic`. The book certainly exposes the fact that the `mm` macros are AT&T's internal macro package which they decided to sell.

I feel that a novice would find Mr. Gehani's chosen order difficult to follow: a very brief history of typesetting, with a description of typographical terms such as point size, fonts and leading, should surely be in the introduction to a book on typesetting? However, his examples of producing letters and memoranda are useful, though the description of producing signature lines comes after the section on headings. The explanation of page headers and footers is after the section on "Interactive Text Insertion": surely this order is wrong? Similarly, in the section on `eqn`, the section on "Labeling Equations" is several pages before the description of producing equation captions.

The idea of setting quizzes at the ends of chapters is a good one, but it is a pity that no answers were provided.

There are very few errors in this book: those which are present are more as a result of ambiguities rather than factual mistakes. However, I found several things to disagree with. For example, Gehani expresses a preference for using in-line escape sequences for font and point size changes (e.g. `\fI`, `\s+1`): his reasons are sound, but this sort of in-line change reduces the readability of the source text for the novice user.

Another failing of Gehani's book is the skimpy explanations of some features. For example, his description of the `troff .ce` command does not include the use of

```
.ce 100
Lots of
text
to be centered
.ce 0
```

to turn centering on and off. The description of user-defined macros does not mention the limitations on the choice of names if the new macros are not to conflict with those used by the `mm` package, and the fact that characters other than "... " can be used to end a macro definition is not mentioned. Further, although Gehani suggests using extra escape characters when referencing a number register as an argument to a macro such as the page header or footer, he does not explain why this is necessary or why a different number of escape characters may be necessary, depending on how often the number register is accessed before it is used. The latter is both simple to explain and important. In short, although this book is aimed at novice users, very little extra effort would have made it much more comprehensive.

I found the section on `troff` disappointing: the descriptions of commands are sketchy. Perhaps this is because they are laid out in tabular form: Mr. Gehani has certainly mastered the `tbl` preprocessor!

As the Writers' Workbench is not available at my site, I am not able to comment on the accuracy of the section describing it. However, if it was used to assist in the writing of this book, it is certainly an excellent piece of software. Mr. Gehani's style is consistent and his spelling is correct (although, of course, American).

Novice users will no doubt find the section of template documents very useful.

One final sour note: I was impressed by the index. Looking in it, I found the entries: "index, generation macros" and "index, template for a book". Eagerly, I turned to page 305. At last, I was going to learn how to automatically produce an index in the approved AT&T manner! Alas, I was disappointed. Although Gehani explains how to write a macro which prints, on standard output, its arguments and the current page number, he glosses over the most difficult parts thus: "combine references to the same item, and further massage the file a bit to improve its appearance and readability". Massage the file? Is this the approved technical term?

I found this book extremely useful. It taught me one thing which is not in the DWB manual, and explained the use of many others in ways which I had not previously considered. I now use the book as a reference in preference to the AT&T manuals, particularly because of its index. I would recommend it as an introduction to `troff` and its preprocessors for someone with a working knowledge of UNIX, at a site with the `mm` macros. It is certainly more readable than the manual!

AUUG

Australian UNIX^{*} systems User Group.

P.O. Box 366, Kensington NSW 2033, Australia.

auug@munnari.oz.au {uunet,mcvax,ukc,nttlab}!munnari!auug

^{*}UNIX is a registered trademark of AT&T in the USA and other countries.

Wednesday 9th December, 1987

John Carey,
AUUGN Editor.

Dear John,

The management committee of AUUG have asked me, some time ago in fact, to write to you, to express our thanks for, and pleasure with the quality and timeliness of the newsletter.

We appreciate the amount of work involved in producing a newsletter of this type, and ensuring a consistent high quality product, and we are grateful that you have been able to do such an excellent job.

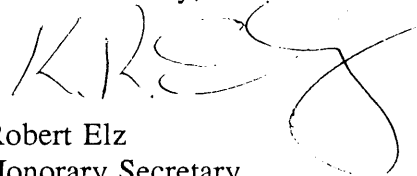
We have a couple of minor suggestions, however, which we believe may help to reduce the cost of the newsletter slightly, while not unduly affecting its quality. We would appreciate it if you would consider these suggestions for future issues of the newsletter.

First, we believe that using a slightly smaller type size might enable more information to be placed on each page, thus reducing the page count. Similarly, putting small items in empty spaces occasionally left at the end of an article might also help.

Secondly, we would request that you take note of AustPost's policy of charging postage at rates that vary in incremental steps, adding just one extra page can vastly increase the cost of mailing an issue. If issues could be planned with these weight steps in mind, perhaps holding some material from one issue for inclusion in the next, we may be able to optimise our use of the post office facilities.

Again, please accept our thanks for the work you have done so far, and our hopes that you will be able to continue to maintain such a high standard.

Yours sincerely,



Robert Elz
Honorary Secretary
AUUG

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 on 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 Memberships are a category that isn't relevant yet. This membership you can't 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. Since AUUG is only just approaching 3 years old, there is no-one eligible for this membership category yet.

Its also possible to subscribe to the newsletter without being an AUUG member. This saves you nothing financially, that is, the subscription price is the same as 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 Mastercard by simply completing the authorisation on the application form.

Robert Elz

AUUG Secretary.

AUUG

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
P O 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.

I, do hereby apply for

- Renewal/New* Membership of the AUUG \$55.00
- Renewal/New* Student Membership \$30.00 (note certification on other side)
- International Surface Mail \$10.00
- International Air Mail \$50.00

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 Mastercard Visa.

Account number: _____ Expiry date: ___/___.

Name on card: _____ Signed: _____

Office use only:

Chq: bank _____ bsb _____ - _____ alc _____ # _____

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

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
P O 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.

..... does hereby apply for

New/Renewal* Institutional Membership of AUUG \$250.00

International Surface Mail \$ 20.00

International Air Mail \$100.00

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 Bankcard Mastercard Visa.

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 usually 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.

- | | |
|---|--|
| <input type="checkbox"/> System V.3 source | <input type="checkbox"/> System V.3 binary |
| <input type="checkbox"/> System V.2 source | <input type="checkbox"/> System V.2 binary |
| <input type="checkbox"/> System V source | <input type="checkbox"/> System V binary |
| <input type="checkbox"/> System III source | <input type="checkbox"/> System III binary |
| <input type="checkbox"/> 4.2 or 4.3 BSD source | |
| <input type="checkbox"/> 4.1 BSD source | |
| <input type="checkbox"/> V7 source | |
| <input type="checkbox"/> Other (Indicate which) | |

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# _____