AUUG

Australian UNIX systems User Group Newsletter

AUUGN

Volume 10, Number 1

February 1989

Registered by Australia Post, Publication Number NBG6524

The Australian UNIX* systems User Group Newsletter

Volume 10 Number 1

February 1989

CONTENTS

* 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,
P.O. Box 366,
Kensington, N.S.W. 2033.
AUSTRALIA

AUUG Executive

Next AUUG Meeting

AUUG89 Conference and Exhibition, will be held at the Sydney Hilton Hotel from Tuesday 8th to Friday 11th August 1989. Further details are provided in future issues.

AUUG Newsletter

Editorial

Welcome to the Newsletter.

As you may be aware I will be finishing my term as Editor following the AUUG89 Conference issue in August this year. The Committee has chosen David Purdue of Labtam Limited to take over as Editor from Volume 10 Number 5. David is already actively involved in the User Group. This includes running the Nutshell Handbook Offer and handling the AUUGN backissues. He has also designed a new front cover which appears on this issue. As you can see he has many talents and I am sure he will add a new dimension to the Newsletter. I wish him every success and urge the membership to support him.

In this Issue there are several informative articles. These include a SunIII starters guide and an introduction to the internet (tcp/ip) protocols which was sent over the net. Mike Karel's "Future Berkeley UNIX developments" slides from his talk given at AUUG88. These have been held over due to various technical difficulties. Regulars such as "Off the Net" and reprints from the USENIX Newsletter also appear.

Thanks to should be given to Robert Elz for getting the gremlins (the drawing program) out of the Karel slides :-)

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

AUUGN Correspondence

All correspondence reguarding the AUUGN should be addressed to:-

John Carey
AUUGN Editor
Webster Computer Corporation
1270 Ferntree Gully Road
Scoresby, Victoria 3179
AUSTRALIA

ACSnet: *john@wcc.oz*

Phone: +61 3 764 1100

Contributions

The Newsletter is published approximately every two months. The deadline for contributions for the next issue is Friday the 14th of April 1989.

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 Tim Roper.

Back Issues

Various back issues of the AUUGN are available on request from the Editor.

Acknowledgement

This Newsletter was produced with the kind assistance and equipment provided by Webster Computer Corporation.

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 Letter

Well, here I am writing this letter in sunny California (except it is raining and about 12 degrees Celcius). Life is certainly busy at the moment, although I'm sure you don't need to hear that.

Just down the road is Ken McDonnell, past holder of this position, but unlike Ken I am just visiting. The good news is that he will be returning to Australia, too.

My sermon today regards doing one's duty, or lack thereof (lack of doing it, I mean). I was just about to set pen to paper and write a president's letter for the last AUUGN when it arrived. I failed to learn from this experience - I should have written it anyway, then I wouldn't be doing it at the last moment, again.

However, one of the primary duties of the AUUG executives, as a whole is to improve the benefits and response to the members, and this is certainly progressing quite well. AUUG Inc has a Convergent Technologies Miniframe, and Informix to run on it, and shortly will be able to process membership requests faster, and even to acknowledge them. (Note: The above resources are courtesy of Sigma Data and Informix, respectively. Many thanks to them).

However, it must be remembered that the committee members are all masochists, I mean volunteers, who also need to hold down jobs to eat. They all do their best, and can use all the help you can give.

Speaking of which, it is nearly time for nominations for the new committee. All members should start thinking about how much better they can do the job, and be prepared to put in nomination forms.

Now for something completely different, organisation for the round of Summer regional meetings was proceeding nicely, until the speaker we had invited pulled out unexpectedly. We'll keep you posted.

Yours faithfully,

Greg Rose,
President, A.U.U.G.

Secretary´s Letter

Firstly, some explanation of the absence of information about the planned informal, technical, summer meetings is in order. A schedule of meetings in seven centres around the country was set for the the period 17th to 24th February, 1989. Local arrangements were well under way in some centres, less so in others. Unfortunately the speaker that we had arranged to import was forced by a change in circumstances to withdraw at the last moment. Since the event had been designed around the one speaker this presented somewhat of a problem. A frantic effort to find a replacement at very short notice was unsuccessful. At its meeting on 3rd February, 1989 the Management Committee resolved to give local organisers the option of postponing or going ahead according to schedule with a local programme supplemented by an interstate speaker provided by AUUG. All organisers elected to postpone. We hope that at least a few of these meetings will still actually occur, with AUUG funding one or two speakers from interstate at each meeting.

On a brighter note, preparations for AUUG89, the 1989 AUUG Conference and Exhibition, are going well. The exhibition space and sponsorships are already heavily booked. If your company is interested in exhibiting its wares or sponsoring an event it should contact ACMS by phone on (02) 332 4622 or fax on (02) 332 4066 immediately. Peter Barnes of the University of Queensland (**pdb@uqcspe.cs.uq.oz**) is Chairing the Programme Committee. A Call for Papers will be issued shortly but don't wait for it before you begin planning your paper. AUUG will be investing more heavily in publicity this year and, with the venue in Sydney, this year's Conference and Exhibition should set new records for attendance. AUUG89 will be held at the Sydney Hilton Hotel from Tuesday 8th to Friday 11th August, 1989, with the Tuesday devoted to tutorials.

AUUG has had an excellent response to its recent Inauugral Software Distribution and Nutshell book offers. If you haven't sent in your order for the Inauugral Software Distribution please do so now as it will be wound down very soon. Such offers usually come about when a member both makes a worthwhile suggestion and volunteers to do the coordination required to carry it through. AUUG has the funds to underwrite these ventures but does not have the fulltime staff to carry them out. So, if you can see a need in the Australian community of UNIX† system users, how about coming up with a proposal for filling it and an offer to coordinate it? With elections coming up in a couple of months time you may like to think about making your contribution by standing for the Management Committee.

Membership numbers are increasing, most notably in the Institutional category (49, see the list elsewhere in this issue). Although the number of Ordinary members is increasing (249) we are aware that we may be losing some who simply who forget to renew. A mailing campaign directed at expired members and meetings attendees is being aimed at signing on some new members and hopefully recovering some old ones. This will become a regular reminder mailing to expiring members. Until then, watch those mailing labels!

Enclosed with the issue is a flyer from the USENIX Association giving details of its *Computing Systems* journal including contents of the first four issues and the half-price affiliated-member rate available to AUUG members. A complimentary copy of Vol 1 No 1 was sent to AUUG members last August. A copy of each issue will continue to be sent to Institutional members, courtesy of AUUG. Vol 1 Nos. 2&3 are currently on their way and we have arranged air shipment to Australia so subsequent issues will arrive in a timely fashion.

Tim Roper

† UNIX is a trademark of Bell Laboratories.

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

Western Australian Unix systems Group

The Western Australian UNIX systems Group (**WAUG**) was formed in late 1984, but floundered until after the 1986 AUUG meeting in Perth. Spurred on by the AUUG publicity and greater commercial interest and acceptability of UNIX systems, the group reformed and has grown to over 70 members, including 16 corporate members.

A major activity of the group are monthly meetings. Invited speakers address the group on topics including new hardware, software packages and technical dissertations. After the meeting, we gather for refreshments, and an opportunity to informally discuss any points of interest. Formal business is kept to a minimum.

Meetings are held on the third Wednesday of each month, at 6pm. The (nominal) venue is "University House" at the University of Western Australia, although this often varies to take advantage of corporate sponsorship and facilities provided by the speakers.

The group also produces a periodic Newsletter, **YAUN** (Yet Another UNIX Newsletter), containing members contributions and extracts from various UNIX Newsletters and extensive network news services. **YAUN** provides members with some of the latest news and information available.

For further information contact the Secretary, Skipton Ryper on (09) 222 1438, or
Glenn Huxtable (glenn@wacsvax.uwa.oz) on (09) 380 2878.

Glenn Huxtable,
Membership Secretary, **WAUG**

AUUG Institutional Members

ACUS / UNISYS
Aldetec Pty Ltd
Australian National University
Australian Nuclear Science & Technology Organisation
Australian Telescope Computer Group (CSIRO)
Autodesk Australia P/L
BHP Melbourne Research Laboratories
CSIRO DIT
CSIRO Division of Manufacturing Technology
Centre for Information Tech & Comms
Civil Aviation Authority
Comperex (NSW) Pty Ltd
Cybergraphic Systems Pty Ltd
DBA Computer Systems Pty Ltd
Department of Industry Technology and Commerce
Dept of Lands - Central Mapping Authority
Elxsi Australia Ltd
Flinders University, Discipline of Computer Science
Fujitsu Australia Limited
Great Barrier Reef Marine Park Authority
Harris & Sutherland Pty Ltd
Honeywell Information Systems
Ipec Transport Group
Lands Department, Qld
Monash University Computer Science
NEC Information Systems Australia Pty Ltd
NSW Parliament
National Engineering Information Services P/L
Olympic Amusements P/L
Overseas Telecommunications Corporation
Prentice Computer Centre
Prime Computer Research & Development
Pyramid Technology Australia
Q. H. Tours Limited
Qld State Govt Computer Centre
Racecourse Totalizators Pty Ltd
Reark Resources
SEQEB
Sigma Data Corporation Pty Ltd
South Australian Institute of Technology
State Library of Tasmania
Sun Microsystems Australia
Swinburne Institute of Technology
University of Adelaide
University of Melbourne
University of Sydney
University of Technology Sydney - Computing Services Division
University of Wollongong
Webster Computer Corporation

ADVANCE ANNOUNCEMENT/CALL FOR PAPERS

International Seminar
on
Current Developments and Future Trends of Unix-based Systems

12 - 13 September, 1989.

organised by
Malaysian Unix Users Association (MALNIX)
in cooperation with
Malaysian Institute of Microelectronic Systems (MIMOS)

OBJECTIVES

To provide a forum for discussion and exchange of informa- tion, ideas and experiences on current developments and future trends of Unix-based systems.

To promote research activities related to Unix in Malaysia and the region.

SEMINAR TOPICS

Parallel Processing	Office Automation
Real-time Applications	Database Systems
Software Development Tools	Networking
Emerging Standards	Unix in Commercial Environments
Unix on PC	User Interfaces
System Security	

SUBMISSION OF PAPERS

A copy of an extended abstract, limited to about 2000 words, should reach the Organiser by April 3, 1989. Notification of acceptance will be sent to the authors by May 22, 1989. The full paper shall be sent, by August 21, 1989, to:

The Organiser,
MALNIX/MIMOS International Seminar,
7th Floor, Bukit Naga Complex,
Jalan Semantan,
50490 Kuala Lumpur, MALAYSIA.

Phone: (60) 3 2552 700 Fax: (60) 3 2552 755
E-mail: malnix@rangkom.my or uunet!mimos!malnix

Nutshell Handbook Offer

Introduction

The "Nutshell Handbook" special offer for AUUG members is going ahead. This article contains a price list and order form; a follow up article will have descriptions of the books available (just in case you missed this before).

I stress again, this offer is for AUUG members only. If you wish to join AUUG, send a message (including your paper mail address) to Tim Roper <timr@labtam.oz>.

I will be placing an order for the books as soon as I receive enough orders. AUUG does not require payment immediately, but will require it before books will be delivered.

Ordering Information

Please complete the attached order form and post it to:

AUUG Nutshell Handbook Offer
Attention: David Purdue
c/o Labtam Limited
43 Malcolm Road
BRAESIDE, VIC, 3195

Payment need not accompany the order form, but books will not be sent until payment has been received (I will post an announcement when the books arrive in Australia). Purchase orders will only be accepted from Institutional members. Others please send cheques payable to AUUG Inc. If your oraganisation is paying, please organise your cheque now.

The prices appear on the order form and include postage and handling charges.

The order form must be signed by a member of AUUG. In the case of an Institutional Member it should either be signed by the Administrative Contact (the person who signed the current membership form) or stamped and signed by a representative of the institution.

Orders will be accepted from non-members only if they are accompanied by a completed membership application form and payment for membership.

Nutshell Handbook Order Form

Contact Details:

Name:

Phone:

Fax:

Net Address:

Postal Address:

Shipping Address (where the books will go):

Books Required:

Title	Copies	Price/copy	Total
Learning The UNIX Operating System		$ 15	$
Learning The Vi Editor		$ 19	$
Termcap And Terminfo		$ 24	$
Programming With Curses		$ 15	$
Managing UUCP and Usenet		$ 24	$
Using UUCP and Usenet		$ 21	$
Managing Projects With Make		$ 15	$
DOS Meets UNIX		$ 19	$
UNIX In A Nutshell (System V edition)		$ 24	$
UNIX In A Nutshell (BSD Edition)		$ 24	$
X Programming Manuals (2 vol set)		$ 89	$
X Window System User's Guide		$ 33	$
Hypercard UNIX In A Nutshell		$ 43	$
MKS Toolkit		$ 124	$
Checking C Programs With lint		$ 18	$
Understanding And Using COFF		$ 24	$
Grand Total			$

Membership Details:

Name of Member (please print):

Category of Membership: Ordinary/Student/Institutional/Hon Life

Signature:

Please note - ADD 20% SALES TAX to prices if applicable - Editor

Nutshell Handbook Summary

This article describes the Nutshell Handbooks being offered for sale by the AUUG. This information is taken mainly from the 'Nutshell News', a press release from the publishers.

UNIX is a trademark of AT&T Bell Laboratories.

Learning The UNIX Operating System

75 pages; AUUG price: $15

For those who are new to UNIX, this will teach just what you need to know to get started and no more. A better introduction than a 600 page book that contains many details irrelevant to the beginner. Topics include logging in and logging out, managing files and directories, redirecting i/o, and customising your account.

Learning The vi Editor

131 pages; AUUG price: $19

vi is the most commonly used text editor on UNIX systems, and is available on other systems aswell. This book teaches how to use vi by starting with basic concepts so that you can begin editing quickly, and then extending your skills so that you can use vi more powerfully.

Termcap And Terminfo

170 pages; AUUG price: $24

The termcap and terminfo databases are UNIX's solution to the dificulty of supporting a wide range of terminals without writing special drivers for each one. Unfortunately, like so many other essential UNIX features, they are poorly documented. This book tries to rectify that situation. Contents include:

- Terminal independence: the need for termcap and terminfo
- Reading termcap and terminfo entries
- Capability syntax
- How users should initialise the terminal environment
- Writing termcap and terminfo entries
- Converting between termcap and terminfo

Programming With curses

71 pages; AUUG price: $15

This book does not tell you what to yell at your terminal when your program won't compile. It does tell you how to use the curses library from C programs to provide a full screen user interface.

Managing UUCP and Usenet

242 pages; AUUG price: $24

This book is meant for system administrators who want to install and manage UUCP and Usenet software. It covers Honey-DanBer UUCP as well as standard Version 2 UUCP, with special notes on Xenix, SunOS and BSD4.3. This book was selected by Usenix's UUNET Communication Services for distribution to new customers.

Using UUCP And Usenet

185 pages; AUUG price: $21

A complete user's guide to UUCP and Usenet, including how to read and post news, send mail to other sites and how to execute commands on remote sytems and log on to remote systems via UUCP.

Manging Projects With make

77 pages; AUUG price: $15

Described as "the clearest book on make ever written" (and from my experience with Nutshell Books, I find that easy to believe). Topics include:

- Writing a simple makefile
- Shell variables
- Internal macros
- Suffix rules
- Special description file targets
- Maintaining libraries
- Invoking make recursively

DOS Meets UNIX

134 pages; AUUG price: $19

There are many applications that run under MS-DOS. UNIX lookalikes (XENIX, Microport, etc.) help you make better use of your PC. This book tells you how to get the best of both worlds. It describes the problems and the solutions available for integrating DOS and UNIX, including a coverage of products like PC-Interface, PC-NFS, Merge/386 and VP/ix that are used to run DOS applications under UNIX.

UNIX In A Nutshell

for System V: 289 pages; AUUG price: $24

for Berkeley: 306 pages; AUUG price: $24

The ultimate UNIX quick reference guides, these contain everything you need to know quickly. The "DEC Professional" (September 1987) states, "I highly recommend the 'UNIX In A Nutshell' handbooks as desktop references. [They] are complete and consise; in fact they pack more information into fewer pages than I've ever seen... These books are truly a bargain." See my review in the December '88 AUUGN. Note that there are different editions for System V and for BSD, so be careful that you order the right one.

X Programming Manuals (2 volume set)

vol1 611 pages, vol2 700 pages; AUUG price: $89

An awful lot of information about X. Volume 1 is a guide to programming with the X library, and takes the broad conceptual view. Volume 2 is a reference manual and gives a detailed description of each of the Xlib functions.

X Window System User's Guide

270 pages; AUUG price: $33

Describes window system concepts and provides a tutorial on the most common client applications. Later chapters describe how to customise the X environment.

Hypercard UNIX In A Nutshell

disk; AUUG price: $43

As for "UNIX In A Nutshell" above, but in the form of a Hypercard stack.

MKS Toolkit

disk; AUUG price: $124

Provides a set of UNIX utilities for DOS systems, including sed, awk, and ksh as a COMMAND.COM replacement.

NEW TITLES

All I know about the following books are their titles and prices:

"Checking C Programs With lint" - AUUG price: $18
"Understanding And Using COFF" - AUUG price: $24

David Purdue
AUUG Nutshell Handbook Coordinator

SALES TAX

Please note - ADD 20% SALES TAX to prices if applicable - Editor

Getting Started and Debugging Guide for SUN III

David F. Davey

Department of Physiology
University of Sydney

ABSTRACT

This document is designed to supplement the guide to the installation of the SUN III software. It details simple checks of the integrity of the installation, and is aimed at trying to catch some of the common problems before they cause trouble. There is also information on some of the "tricks" that can be used to detect and work around faults in the network operation.

1. INTRODUCTION[1]

This document is designed to assist in:

- getting a new installation running even if you know little about SUN III
- solving some common problems with running installations

The document begins with consideration of some important decisions that must be made before your first network connection is established, some of which are difficult to reverse. This is followed by instructions on how to perform some tests before you attempt to make contact with your network "neighbour(s)", i.e. before starting the daemon(s). You can be sure that if these tests fail without the daemon(s) running, things will not be improved by starting them.

1.1 Preliminaries – node and domain names

There are some decisions you must make before your node can make contact with the rest of the network. These include: what node your node will link to; what name your node will be known by; and what domains your node will belong to, including probably a domain you create for your site. Your link arrangements must be made with the cooperation of the site you wish to link to.[2] Contact with the system administrator there is absolutely essential, and in the first instance must be by non-network means. This person may also serve to advise you with regard to the name questions below, either to answer questions or refer you to someone who can. You should already have read the relevant parts of the *Installation Guide* and the document on *Domain Addressing on SUN III*, but some important points will be reiterated here:

- Your domain name, assuming you create one for your site, *must* be unique within the regional domain to which you connect.
- Your node name *must* be unique within any domain you are a member of.
- Your node name is best chosen to be unique within SUN III.
- There are some advantages to a node name which is absolutely unique.

1. Although I *initiated* this document, partly because of experiences related to aus.map (see AUUGN 9(1):1-13), it would not be in its present form without substantial help from Robert Elz, and additions and improvements from John Mackin, Stephen Frede and Chris Maltby. Moreover it would not exist at all without Piers Lauder, not only because he is responsible for SUN III, but because he tolerated what must have seemed an endless series of questions. Suggestions for improvements or additions to daved@physiol.su.oz please.

2. If you need to find a site to connect to, there are some sites that will provide a link for a fee, e.g. metro.ucc.su.oz (University of Sydney Computing Service).

- You should avoid names with equivocal characters (e.g. vax1 *vs.* vaxl). Such names *will* be confused.
- Be creative when deciding upon your node name.[3]

Be Warned: It is difficult to change your node name.[4] It is close to impossible to change your hierarchy. Therefore it is important to get these right at the outset.

1.2 Preliminaries – The SUN III account name ACSNETNAME

As stated in the *Installation Guide* the SUN III processes will run under the account name **ACSNETNAME** and its associated **ACSNETUID**, and is installed under group **ACSNETGROUP** and its associated **ACSNETGID**. There are a number of advantages in creating a special account for this purpose. Indeed on some systems, notably those that restrict the number of processes that an account can run at one time,[5] there are dangers in not doing so, for at times SUN III can generate a substantial number of processes. On systems supporting disc quotas, having a separate account for SUN III can be useful in keeping track of disc usage. On systems with more elaborate accounting, other parameters may be monitored. It is also possible to allow certain users to know the account password in order to allow access to SUN III files that would otherwise be restricted, although on some systems this can be managed more easily through **ACSNETGROUP** access.

1.3 Preliminaries – the compilation configuration

As described in the *Installation Guide*, there are two ways to organise the compilation of SUN III for your system: to edit the template Makefile (*Makefile.dist*), or use a *run* file. The latter method is preferable because it is much easier to adapt to a new distribution of the software.[6] If your system is one already catered for in the distributed *run* files (in the *Makefiles* directory), your task should be easy. Nevertheless your *run* file must be checked carefully and will almost certainly need to be edited with regard to simple matters like **ACSNETNAME**, **ACSNETUID**, and **FUZZ**, with careful reference to the *Installation Guide* for the definitions. The *run* files also tend to get out-of-date, so it is also important to look at the *Makefile.dist* that came with your distribution and check that each parameter will be set the way you want it, adjusting the *run* file where necessary.

If your system is one that "auto-nices" long running processes it is essential to set **NICEDAEMON** to a negative value, otherwise the daemons will become so "nice" they will cease to be of any use.

If you are configuring for a system not referenced in any of the *run* files, or for a new CPU, particular care must be exercised. Examination of values for **CONFIG** and **CFLAGS** in the available *run* files will give hints of places to look out for problems. If you do configure for a new system, share the result with future installers by sending the working *run* file to wherever you obtained your SUN III distribution.[7] If you are aware of deficiencies in your compiler, e.g. with the optimiser, adjust **CFLAGS** accordingly, as optimisation is used by default.

For any "first-time" installation, or one which is not on a fairly standard system, it is worth setting **DEBUG**=2 so that tracing can be enabled if any debugging is required.

3. Your node will be known by its name. People will use the name in speech and text. It is nice if the name is pronounceable. It is boring if it is yet another combination of meaningless letters coupled to a machine type (e.g. *yavax*). Besides when you replace your Vax™ with a Cray™, it will be silly for it to be called *yavax*.
4. Strictly this means changing your node name is likely to have so many adverse effects that doing so is masochistic even though the act of changing the name is easy. If you have a network of Suns, be aware that randomly changing your hostname or domainname (with those commands) may interfere with NFS, YP etc. Set these names first, before compiling SUN III.
5. Notably 4.3BSD and derived systems.
6. New distributions will be necessary as new features are added to the network and bugs are eliminated. A sound *run* file can make the installation of such updates a trivial task.
7. Send it to piers@basser.cs.su.oz by default.

1.4 The compilation

Assuming you have your *run* file ready, the basic steps in the compilation and installation are just:

```
./run certain
./run directories special
./run install
```

2. GETTING STARTED

2.1 Tests of user commands to perform before starting the daemon(s)

Once the SUN III software is compiled and installed, there are a number of tests you can perform to help ensure the integrity of the installation which do not require the network daemons to be active, i.e. these tests can be (or should be) carried out before you attempt to make contact with other machines. These tests are best performed when logged in *without* special privileges, to check that the correct permissions have been established for ordinary users.[8] None of the tests should result in error messages if the installation is sound. If error messages do occur, hopefully they will be self explanatory, and the fault should be corrected; some of the possible causes of faults are detailed with each test.

- Try sending yourself a file:

  ```
  sendfile[9] my_login file_name
  ```

  There should not be any error messages (unless you use a non-existent login name or file name). You should receive mail telling you the file is available, and then, as described below, you should be able to collect the spooled file.

- First examine the mail. Check the headers carefully to see if the addresses are correct (the mail interface is a common problem with new installations).[10]

- Try collecting the file. It is worth changing directory first so collecting it will not interfere with the original

  ```
  cd /tmp
  getfile
  ```

  You should need only to type a 'y' and a RETURN in response to the query. Examine the file ownership and mode after collection. With the possible exception of permissions masked by your *umask* they should be the same as the original. Compare the collected and sent file, which should be identical.

---

8. However if your installation makes use of flags to control network access (e.g. AUSAS or MUSH), make sure the account that will do the testing has the appropriate flag(s) first!

9. The names by which the SUN III commands are known are configurable. The names used in the printing of this document are related to the names used in the configuration as follows:

CON	`con`
FETCH	`fetchfile`
FILEGETTER	`getfile`
QUEUE	`acsqueue`
SEND	`sendfile`
STATEP	`acsstate`
STOP	`acsstop`
WHOIS	`acswhois`

10. If mail advising you that the file is available for collection is not received, see the section "PROBLEMS WITH THE MAIL INTERFACE" for help on specific problems. It cannot be over-emphasised that lack of mail at this stage is a serious fault, since almost all the SUN III error handling is by mail to the account set with **FUZZ** in the compilation. If such mail cannot be delivered, faults will never be apparent.

- Check that mail to your network address works:

  ```
  mail my_login@our_node
  ```

  Again check the headers carefully, paying particular attention to the "From " and "From: " lines which should contain your user name and node name in the form *username@node.domains*. If the node and/or domain information is missing, it could be for any one of a number of reasons, some of which do not indicate faults. Nevertheless it is worth checking this out by referring to the section on "Testing network mail locally".

- Check that the commandsfile information has correctly set up the statefile:

  ```
  acsstate -V our_node
  ```

  The output should look something like:

  ```
  our_node          {domain1.domain2.oz} [domain1|domain2]
                     "Our comment string"
  ```

  This fictitious output would relate to lines in the commandsfile:

  ```
  #
  # configure our node
  #
  domain      our_node           domain1,domain2
  hierarchy   our_node           domain1.domain2.oz
  comment     our_node           "Our comment string"
  ```

  In other words, the output should show an entry for your node name, with its hierarchy in braces and its domains in brackets separated by '|' characters with the primary domain listed first.[11] Your comment should follow. With the possible exception of the comment, all these components should be present in the acsstate output. The hierarchy *must* end in the top domain, currently "oz" for Australia.[12] If anything is missing from the output, you should edit the commandsfile with reference to the *Installation Guide* and the manual for acsstate; a task which must be performed with system privileges.[13] There could be other things listed if you put them in the commandsfile, such as "caller" or "filter".[14] If you put information on links in the commandsfile (not the recommended procedure), information on these links will follow the acsstate output on "our_node". Assuming you did not put the link information in the commandsfile (the recommended procedure), some of the simple tests below will not work, so you should temporarily get acsstate to add information to the state and routing files for the node to which you intend to link, which is called "neighbour" in this document. Execute the command:

  ```
  acsstate -RSWC<<![15]
  add         neighbour
  link        our_node,neighbour    msg
  !
  ```

11. The node you intend to connect to must be a member (at least) of this primary domain.
12. Some other top domains are "nz" for New Zealand, "th" for Thailand.
13. Whenever you edit the commandsfile you should run the command:

    ```
    acsstate -RSWc
    ```

    which will force the changes into the routing file (–R) and the state file (–S), as well as give warnings (–W) about any faults. The c argument must come last to indicate the default commandsfile.
14. It is worth noting that the contents of the commandsfile should be kept to a minimum and managed very carefully (see section "ROUTING PROBLEMS".)

This will have the side effect of generating a state message to inform the network of your new link to "neighbour".

- Check that the link information is correctly displayed by `acsstate`.

  ```
  acsstate -V our_node
  ```

  The output should now look something like:

  ```
  our_node           {domain1.domain2.oz} [domain1|domain2]
                      "Our comment string"
   ->           neighbour (msg)
  ```

  i.e. the link to neighbour is shown (indicated by "->"), together with the link flags in parentheses.

- Check that the link(s) you expect to have are listed by the `linkstats` command. Try

  ```
  linkstats
  ```

  which should print a header, then one line of output for each link you intend to have. The absence of such lines most likely indicates that the directories for them have not been set up. See the *Installation Guide* "Starting the Network".

- Try to send a file to your nearest neighbour

  ```
  sendfile root@neighbour < /dev/null
  ```
  [16]

- Now check that this file has been spooled for transmission (it will not actually be transmitted until the network daemon is active):

  ```
  acsqueue -V neighbour
  ```

  which should give output something like:

  ```
  neighbour         daemon inactive
                    2 messages in queue
  2: Files from my_login at our_node.domains to root@neighbour at neighbour 138 bytes
                   0  Jan 17 12:04  stdin
  ```

  where the number of bytes will probably be different, and the time is the last modification time of /dev/null, not when the message was sent. The unlisted first message will be the state message generated above.

- You probably do not want this test message to be delivered, so check that you can stop it:

  ```
  acsstop
  ```

  which should result in output something like:

  ```
  Link to neighbour:-
  Files    from my_login    to root@neighbour at neighbour    138 bytes
  Stop ? (y or n)
  ```

  To which you need type only a 'y' and RETURN, after which `acsstop` should report:

  ```
  1 message stopped.
  ```

15. The upper case C tells state to read commands from standard input instead of from the commandsfile as in the previous example.

16. The use of `sendfile` redirected from /dev/null is the most efficient means of testing `sendfile`, as no file to send is needed, and a zero-length file is sent.

which you can confirm with the `acsqueue` command.

- You can check the `acswhois` command:

```
acswhois my_login@our_node
```

  which at the very worst should give you the message "Information not available", indicating the whois database you specified in the configuration is not available, possibly by choice.

- You can check that the `fetchfile` command is operational:

```
fetchfile -dour_node -L .
```
[17]

  What will happen is dependent upon whether you configured your installation to serve as a fileserver host[18] by defining **PUBLICFILES** [19] in your configuration. If you did, the `fetchfile` command should result in a file called PublicFileList being sent to you. The message:

```
fileserver: error --  No public files are currently available.
```

  indicates the directory you specified in the configuration is not accessible. If you did not define **PUBLICFILES** you should get the message:

```
There are no remotely accessible files at our_node.  Sorry.
```

2.2 Tests of system commands to do before you start the daemons

These commands can only be executed with system privileges. Some of them are not installed in your system bin directory, but are to be found in the _lib directory in the SUN III spool directory.

- The `purge` command should be run regularly (see "Maintenance" section) to clean up the SUN III directories of out-of-date files. You should check that this command is executable and does not report any errors

```
./purge -W
```

- Many of the standard user commands have flags that only the system administrator can use. e.g the `getfile` command can be used to collect (or delete) files sent to any user. You should test that one of the restricted flags will work, e.g. try:

```
getfile -LA
```

  which will list all files spooled regardless of who they were sent to. This is valuable in checking for uncollected files, or in cleaning up if disc space is short.

- The `request` command is one you will probably not need to use, and certainly should not use unnecessarily.[20] Nevertheless, in a sound installation it should work, so try this simple test of the `request` command:

```
cd /usr/spool/ACSnet/_lib
./request neighbour
```
[21]

---

17. Note that the last argument is a dot.

18. Acting as a fileserver host means that you have a place where you put files that users at other nodes may request to have sent to them. If you think that your site may be a source of files for other sites, you should define **PUBLICFILES** in your configuration. If you want to minimise the size of your SUN III installation or exclude the possibility of a security hole that acting as a fileserver could represent, leave it out. (Note: this is not to suggest there are *known* security problems with `fetchfile`.) Consult the *Installation Guide* for more information.

19. Upper case names printed in this way refer to compile time definitions which are site configurable. They are described in the *Installation Guide*.

20. Amongst other things, reverse charging applies to the file(s) the request causes to be sent to you.

This command should queue a state file for delivery to neighbour.

```
acsqueue -VMA
```

should list this message; the listing should include the string "ENQ"

2.3 Simple tests of the mail interface

First check that mail to a non-existent node generates an error.

```
mail blogs@notanode
```

should, depending on your *mail* program, either give the message:

```
sendfile: error -- "notanode" unknown
```

or generate mail to you informing you that the mail was undeliverable.

Check that mail to a user at a node that is in your statefile ("neighbour" in the examples above), is spooled for delivery.

```
mail blogs@neighbour
```

should *not* generate any errors, and it should be visible with the `acsqueue` command. The command

```
acsqueue -VM
```

can be used to check that the message has been queued, but also that the "mail envelope" is correct, i.e. that the sender and destination look correct in the `acsqueue` output. (Note that since these are generated by the *mail* program initially and not by SUN III, we are checking something quite different to the `sendfile` tests above.)

Next check that the contents of the message are correct. Change directory to **SPOOLDIR**/neighbour and list the directory contents. There should be a file with a name made up of apparently meaningless characters; This file contains information as to where the files containing the message and the SUN III message header are located. Use *strings(1)*[22] to examine this file. The file name in the _work directory is the critical one. It is essentially a binary file, although in many cases it may begin with an ascii component. It will always end with the binary SUN III FTP. Examine this file to see if the mail header lines look correct – again *strings(1)* or a similar program can be used. The command

```
sed "" spool_file
```

is a simple way to strip the FTP from an otherwise ascii file.

If the file contents do not appear sound, consult the section "PROBLEMS WITH THE MAIL INTERFACE". You should now `acsstop` the message.

2.4 Tests on network-related commands not really part of SUN III

The internationally agreed upon standard account for mail enquiries at all sites is "postmaster"[23]. You should check that

```
mail postmaster
```

works and that the mail is delivered to someone who will attend to it. If your system does not already

21. on a "standard" installation. Your _lib directory could be somewhere else according to your configuration at compile time.

22. In its absence use `cat -v` or `od -c`.

23. See RFC822 "Standard for the format of ARPA Internet messages". This document is obtainable from a number of sites, including munnari.cs.mu.oz and physiol.physiol.su.oz, e.g.

```
fetchfile -dphysiol.physiol.su.oz rfc/rfc822
```

have a postmaster facility, you should establish it. If your mail programs support aliases for incoming mail you can alias postmaster to root or someone appropriate. If it does not, you must create an account postmaster, preferably arranging for mail forwarding to the system administrator. Ideally, all case variant names should be recognised (i.e. Postmaster, POSTMASTER, PostMaster etc.) If your mail program accepts case independent names, such variants will obviously be accepted, but if not, the obvious additional aliases should be included.

2.5 Testing a daemon and a link

There is really only one test of what is happening on a link: the `linkstats` command. It can be given a flag (–c) to repeatedly output the status of the link, and this output can be piped to the command `dis`.[24] If your system supports windows, or if you can devote a terminal to this command, run:

```
     linkstats -Vc3 neighbour | dis
```
[25]

while the daemon is starting, and while you try to establish your link and perform some of the operations in the section "ESTABLISHING YOUR LINK".

2.6 Cleaning up after the tests

Any test files spooled for link "neighbour" following the above tests should be removed, and if you added link information by reading from standard input as recommended above, this link information should now be removed:

```
     acsstop -AY neighbour
     acsstate -RSWC<<!
     remove neighbour
     !
```

3. ESTABLISHING YOUR LINK

There are a variety of ways your node may establish contact with another node. The nature of the connections involves two issues: the nature of the physical link; the nature of the logical link used over the physical link. You may have a dedicated connection between your node and the one you link to, e.g. twisted pairs and line drivers between two machines in reasonable proximity, or an Ethernet, or even an infrared or microwave link over longer distances. For isolated sites it is more common to make use of a telephone modem link, usually on an intermittent basis, especially if the telephone calls involve long distance charges. Over dedicated links it is common to have node-to-node communication operating on a permanent basis. This minimises transmission delays. However, sometimes even a dedicated link may be used for other purposes, e.g. remote logins, and can be used for SUN III communication only intermittently, although it is possible to do both if you can run a multiplexer protocol over the link. (See "Remote logins using SUN III".)

Intermittent links have a calling and a called end. You may wish to provide for both if possible.

24. The source for `dis` is distributed with SUN III, but it is not installed as a part of the normal installation. To compile it:

```
     cd <your SUN III source directory>
     cd Control
     make dis
```

If you install this command, you should also install the manual distributed with the sources.

25. Some administrators find a shell script or function to do this is handy:

```
     linkstats -Vc3 ${1:-neighbour} | dis
```

3.1 Common requirements for all links

There must be a directory with the SUN III spool directory, **SPOOLDIR**, named with the name of the link. This directory must be owned by **ACSNETNAME**, and be read-write-executable by owner. Unless you are paranoid, it can be read-executable by mortals.

If there are special reasons to do with the nature of the link, you might need to insert appropriate "add" and "link" lines to your commandsfile with a minimum of detail about the link.[26]

Within the link directory, you should create a file called *params* which can contain arguments for the daemon used on the link. It can be empty at this stage, but should be readable by **ACSNETNAME**.[27]

3.2 Permanent links over dedicated connections

This is probably the simplest form of link. There must be a special file which the daemon will use to connect to the remote link. This file must therefore be appropriate to the physical connection to be used, and must be read-writeable by **ACSNETNAME**. The file name can be passed to the daemon as a –d*special_file_name* argument, or the default name can be used which requires that in the directory /dev/net there be a file having the same name as the link's nodename, e.g. for a link to node "neighbour", the file */dev/net/neighbour*. Unless the transmission speed of the link is hard wired, edit the link *params* file to contain a –p*speed*[28] flag according to the baud rate the link will be run at (remembering that this will have to be the same at both ends). All that needs to be done to activate the link is to run the NNdaemons at both ends, assuming that the physical link is sound[29]. These should normally be started in the system startup code, e.g. by placing an appropriate line in */etc/rc* (or the like):

```
(cd /usr/spool/ACSnet/_lib; ./rundaemon -I -w60 neighbour)
```

The command `rundaemon` is used instead of directly invoking `NNdaemon` because the NNdaemons have a nasty habit of dying from time to time and rundaemon will start a new daemon upon such a death. Alternatively, if your system supports /etc/inittab, the NNdaemon can be spawned from it (with a respawn flag). Once the NNdaemons are running at both ends, `linkstats` should show the link as up and `acsstate` should show that there has been an exchange of state information. (See "Testing a daemon and link.") Any failure at this stage is most likely due to incorrect special files, mismatched speeds or problems with the physical link such as incorrect cabling.

3.3 Setting up for incoming calls – intermittent links

If you are arranging for an intermittent link, most commonly over modems, setting up for the remote site to call you in the first instance is easiest. The daemons are started through the remote system logging into yours, so whatever the incoming route, it must be set up to accept logins.

- Make an account for the calling node. The account name should be the name of the calling node.[30]

26. There are pros and cons to doing this. The state information concerning the link will be established automatically when SUN III daemons first run on the link, and thus it is not necessary to include link information in the commandsfile. Indeed the "add" command overrides SUN III's automatic configuration behaviour. On the other hand, it is not possible to spool files for a link until the statefile contains information about it. This makes preliminary testing somewhat difficult, but can cause serious problems if the statefile is corrupted or has to be truncated, and the only state information will come from the commandsfile.

27. If you intend to let the remote site be responsible for the link, the file should be owned by the user from the remote system who must be able to adjust the contents to match those at the remote end.

28. Note that the *speed* is not the baud rate *per se* but the system define for that speed. See the *NNdaemon(1)* manual.

29. One way to test the physical link is to get the remote site to run a getty on their end of the line, and use a utility like *cu(1)/tip(1)* or something similar (SUN III'c `con` utility for example) to connect to the line at the local end and make sure you can log in at the remote end, and that you can examine large files on the remote machine without character loss or corruption. (See the section "Remote logins using SUN III" for information on how to use `con` for this purpose.)

30. If you cannot use the nodename, e.g. if it has more characters than your system will tolerate for login names, or if the login name is already in use, chose a reasonable alias that you can use for a login name, and create an account with this name instead. Then add the line

```
alias          real_nodename          chosen_alias
```

to your commandsfile after the "link" line for the node. Finally advise the calling node that they will have to use the alias you

(It will be used by the calling program at the remote site.)

- Set a password for the account (which you will have to agree upon with the administrators of the calling node; it will be used by the the calling process to effect the login). If your system supports password ageing, disable it for the account.
- Set the home directory of the account to be **SPOOLDIR**/nodename.[31]
- If your SUN III installation supports any accounting system to permit/deny network access, give the necessary network flags to this account. If your login procedure allows control of logins over dialup lines, and one is to be used, be sure the account has the necessary flags.
- Set the login shell of the account to be **SPOOLDIR**/_lib/NNshell[32]
- Within the link's directory in **SPOOLDIR**, create a directory called *q* with the same ownership and permissions as the link directory (see above). This step is optional but generally advised (see the NNdaemon manual explanation of –r).
- Create a *params* file which at this stage need only contain the line

  ```
  -rq
  ```

  if you are going to use the *q* directory created above.

Getting the link going will now be largely the responsibility of the calling site. If problems arise, addition of –T1 to the *params* file will assist in diagnosis (with the output appearing in the *log* file). Selecting seven bit mode (–C in the *params* file), or changing the speed of the line, will have to be done with the guidance of the calling node, for these parameters must be changed at both ends.

3.4 Setting up for outgoing calls – modem links

Many of the requirements for initiating calls to another site are the same as for receiving calls, and the details of these can be found above: a link directory within **SPOOLDIR**; a *q* directory within the link directory; a *params* file (probably containing –rq).

In addition you need a calling program. This program must know how to initiate calls on your modem and how to log in at the remote site. If you have a common modem, and the remote login procedure is standard, all you need do is compile the appropriate call program from the distributed sources in the NNcall/Callers directory, and install the binary in your **SPOOLDIR**/_lib directory.[33] If you have an unusual situation, and cannot find a site with similar demands, you will have to create your own caller, probably by starting with a common one like hayesmodem.c. Now link the call program binary in _lib to *call* in the link's directory. Create the file *callargs* and place in it the appropriate arguments for the calling process, i.e. giving *call* the number to telephone, the name to log in under, the password etc.[34] Note that this file should probably *not be readable by mortals* since it contains the remote password and

---

selected.

31. This is not essential, but it can be convenient to be able to

    ```
    cd ~nodename
    ```

    to get to the spool directory.

32. On some systems, where timezones are set from /etc/profile, you might change the initial shell to be **SPOOLDIR**/_lib/netlogin which would be a shell script that set the timezone (/etc/TIMEZONE perhaps) and then executes NNshell "exec **SPOOLDIR**/_lib/NNshell".

33. See the *Installation Guide* and the NNCall Makefile to see how to do the compilation.

34. If this seems a little vague, it is because the call arguments *will* vary substantially with the call program (not even the flag letters are consistent). For the hayesmodem caller it might look like:

    ```
    -d/dev/modem -s2400 -pP12345678 -llogin -Psomepasswd remote_node_name
    ```

telephone number. To test a call to node "neighbour", `cd` to your **SPOOLDIR** and run

```
    _lib/NNcall -T1 -hneighbour neighbour/call
```

On intermittent links it is often necessary to run the daemon twice to make anything much happen. The first time allows the remote node to know you exist, which starts it sending state information everywhere, including to you. Usually the daemons have seen that there is nothing in the link queues well before that happens, and so they quit. A delay of a few minutes between the first successful call and the second one is a good idea, thereby making sure the daemons have sufficient time to queue their state messages, unless the commandsfile contains explicit link information in which case the state messages should have been queued and transferred. In between the two calls, the calling site can use `linkstats` to see if a message was transmitted. If so, it was your state message going out; if not, then it should appear in the outgoing queue, and can checked with the

```
    acsqueue -AVM
```

command.

If the daemons start, indicating that the connection was made, but die, then something is probably wrong. First make sure the problem is not just that the remote site is very busy; add a -B*N* flag to your *neighbour/params* file to increase the delay before the daemon will time out. *N* is 11 seconds by default. If this still does not work, the _bad directory and log file should be checked, and core dumps sought in the obvious places (e.g. **SPOOLDIR**, **SPOOLDIR**/neighbour). If the daemons run for a reasonably long time, then die, check that you do not have a problem with "auto-nicing" (see the "Preliminaries – the SUN III configuration" section).

Once calls can be made reliably, you should organise to make them on a regular basis, invoked by *cron(1)*,[35] at times agreed upon at both ends of the link.

3.5 TCP/IP links

This protocol essentially involves a call operation. For an Ethernet and 4.3BSD, ENcall and ENshell programs exist to establish these links. If you are setting up a new Ethernet, you will almost certainly need advice from a site that has a similar operating system and a working system.[36] If you are just adding a node, seek advice from elsewhere on the TCP net.[37]

3.6 X.25 links

Another call type link. Seek help from the other end.[38]

3.7 Tests of commands once the daemons are running

There are some obvious tests to ensure everything is working. Once the daemons have made contact, they should immediately exchange network state information. This means the bare bones statefile

35. A typical *crontab* entry might be

```
    15  4,12,16,22 * * 1-5 /bin/sh SPOOLDIR/netcall neighbour
```

where *netcall* is a shell script to invoke `NNcall` containing essentially:

```
    cd SPOOLDIR
    HOST="${1:-neighbour}"
    _lib/NNcall "-&" "-h$HOST" "$HOST/call"
```

but which could have local features such as means of correctly setting the time zone if necessary.

36. cad.eecs.unsw.oz and physiol.physiol.su.oz have 4.3BSD Ethernet systems in use.

37. The problems with these links can be very subtle; e.g. the inetd configuration on BSD can, if wrong, cause faults that appear to be due to SUN III.

38. All these links have unique features that makes generalisation impossible.

created from the information in your commandsfile should be expanded. That this is so can be tested with

```
acsstate -V
```

which should now give more information. There will probably be many more nodes unless your primary domain is very restricted. At the very least there should be more information on the node to which the daemon is communicating, e.g. its comment.

A simple test message is a good idea:

```
sendfile -A root@neighbour < /dev/null
```

This zero-length file should be transmitted and acknowledged (because of the –A flag) very quickly.

You can test that you can send a file to yourself, routed *via* you neighbour:

```
sendfile my_login@neighbour!our_node < /dev/null
```
[39]

Once again this message should be transmitted and returned quite quickly.

You should also test that a message sent to a non-existent user at your node results in mail to you advising of the failure to deliver, and to the **FUZZ**, if you have **LOG_RETURNED**=1 in your configuration

```
sendfile bogus_login@neighbour!our_node < /dev/null
```

The undeliverable message should be saved in the _bad directory within the SUN III spool directory. The mail to the **FUZZ**, if it exists, should refer to this file. Check that the _bad file has been created; then you can remove it.

3.8 Making yourself known

Once you are satisfied that your configuration is sound, you should arrange for your site information to be entered in the "Network Map" which is distributed by the network news system both over SUN III and overseas. To acquire information about how to do this:

```
fetchfile -dphysiol.su.oz README acsmap_form
```

which should result in the delivery to you of the two named files. If this does not happen within a reasonable period, there is a potential that something is wrong with your SUN III configuration. You should send mail to *acsmap@physiol.su.oz* describing what has happened. The map moderator will try to help ensure your configuration is sound and that messages can be routed to you.

4. REMOTE LOGINS WITH SUN III

The `con` utility distributed with SUN III allows you to logically connect your terminal to any serial device your hardware and operating system supports. Usually the serial device is a dedicated link to another machine with a *getty* running on the remote end.[40] SUN III supports routing to remote machines through a series of `con` processes, although it is mandatory that the *getty* at each node can fork `con` to connect to the next link in the chain of `con` processes.[41]

39. The syntax destination_1!destination_2 is called *explicit routing*. The message will first be routed to destination_1, and when it arrives there will be routed to destination_2. These destinations need not be directly linked. N.B. If your shell is the csh or any other that treats the '!' character specially, you will have to escape it.

40. If your system remote login procedures such as *telnet*, `con` may seem somewhat superfluous. However if you have a number of machines not all of which support *telnet*, `con` may assist you in connecting to such machines. Furthermore, `con` can provide a redundant connection mechanism between machines linked by other means to cope with hardware or software failures.

41. This system is used extensively at the University of Sydney and University of New South Wales on a variety of UNIX versions.

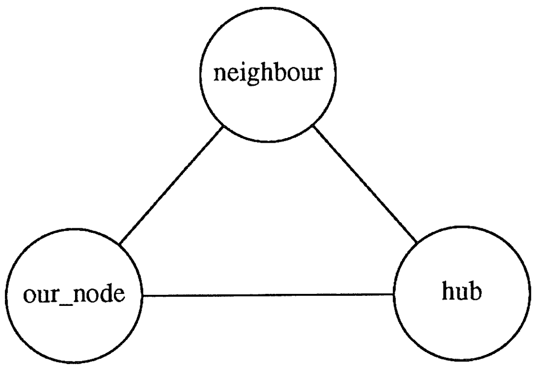
To support `con` to a directly connected node "neighbour" over a serial connection, there must be a device special file */dev/net/neighbourØ*. (Note that the 'Ø' is obligatory and that the file is very different to */dev/net/neighbour* (see "Permanent links over dedicated connections"). Generally the file is a link to the conventional *tty* device special file in use for the serial connection.[42] If more than one `con` channel is provided, additional sequential device special files must be created, e.g. */dev/net/neighbour1* etc.,

4.1 `con` in a local network

If a number of machines are linked with `con` support, SUN III will route `con` requests according to state information, provided the commandsfile contains appropriate information, e.g. if your machine is linked to node "neighbour" and supports both NNdaemons and connections, your commandsfile "link" or "flag" line would look like:

```
link          our_node,neighbour    msg,con
```

If `con` is used extensively, all available connections can be busy when a user tries use `con`. If there is an alternative route to the target, SUN III *will not* use it if the lines are busy. There is a trick to exploit such alternative routes. Consider the following simple network:



Suppose you have two `con` channels between each pair of machines. The first two users of `con` from "our_node" to "neighbour" will use */dev/net/neighbourØ* and */dev/net/neighbour1* respectively. A third user's attempt to `con` to "neighbour" will fail and SUN III will *not* exploit the available route to "neighbour" *via* "hub". However if you create a link between the device special file */dev/net/hubØ* and */dev/net/neighbour2*, the third `con` attempt will actually go to "hub" which will in turn connect to "neighbour".[43]

4.2 `con` as a general testing utility

To use `con` to connect to any serial line, e.g. a modem, simply create a link to the appropriate device special file in /dev/net, e.g.

```
ln /dev/modem /dev/net/modemØ
con modem
```

If the hardware requires that carrier be asserted before the device can be opened, `con` will not work without it.

42. In the case of multiplexed links, it would be a link to the device special file for the appropriate channel on the multiplexed port.

43. Assuming the *getty* running on "hub" knows how to fork `con`. If you have the sources to the login process adding support for this is quite easy.

5. MAINTENANCE

Most SUN III maintenance can be automated. The following recommendations are not essential if your administrators are willing to do things by hand, but are *highly recommended* to make your installation as trouble free as possible in the long term.

- `purge` should be run regularly – every night at most sites, more often at busy sites – to remove out-of-date state messages[44]. This may mean adding an entry to the *cron(1)* table, or adding it to a script run for related purposes (e.g. /usr/adm/daily on BSD systems). The output generated by the –W flag will alert you to potential faults, so redirecting the output of `purge` to mail to the system administrator is worth considering.
- Remove (or otherwise deal with) old files in the _files directory. When users are advised of the availability of files for collection with `getfile`, they are told they should collect or delete them within **FILESEXPIREDAYS** or they will be deleted. This deletion will not occur unless you arrange to do it. If you want to be rigourous about this, simply remove, on a daily basis, any file in the _files directory older than **FILESEXPIREDAYS** relative to the current date. This can be done using *find(1)* or

  `getfile -AoN -delete`

  where *N* is **FILESEXPIREDAYS**, or perhaps some slightly larger number to give a period of grace.

  On the other hand, if you have users who log in rather infrequently, and/or you wish not to delete potentially valuable files, you may choose to generate further messages once a certain period has passed, and perhaps to later alert the system administrator.[45]
- Truncate the log files in each link directory. The strategy needed varies greatly with the nature of the link. The log file for a dedicated and reliable link will not grow at a great rate, and might need to be truncated only rarely.[46] A log file for an intermittent link with frequent calls and/or a link with frequent faults can cause the log file to grow rapidly, its truncation might need to be done on a daily basis.
- Discard old state files in the _state directory. Any files here older than a few months are of doubtful value. They should be removed, perhaps weekly, using find.[47]
- Examine then discard state [illegible] messages in _state.[48] These files have the same names as the state files to which they [illegible] but with a ',' prepended to the name. The contents may give important information about state inconsistencies, especially those between incoming state messages and the contents of your statefile. Even if you choose to ignore these messages, you should truncate the files which can otherwise grow indefinitely.
- Examine any files in the _bad directory. If there are files here, they are a result of some failure of SUN III to deliver messages - e.g. messages to non-existent users, messages that have been looping around a route, or messages saved because some process forked by SUN III failed, e.g. news. The script badmv.sh distributed in the SUN III sources Admin directory can be used to deal with such

44. See the `purge` manual for more details.
45. The shellscript oldfilewarn.sh performs this task.
46. For example only when the system is rebooted. The system startup file (/etc/rc or the like) might contain, for a link called ***neighbour***:

```
cd /usr/spool/ACSnet/neighbour
cp log OLDlog
> log
```

47. The script purge.sh does the double duty of running the `purge` command and discarding old state files.
48. The shellscript staterep.sh will do this for you.

files.

6. PROBLEMS WITH THE MAIL INTERFACE

As many of the problems to do with SUN III relate to its use with electronic mail, considerable attention to certain problems is given below. It is important to appreciate that the Mail Interface is complex, and really consists of several parts:

1. messages pass from your user mail program(s) to SUN III by invoking `sendfile`
2. SUN III messages destined for your mail system are passed to **MAILER** with arguments selected by **MAILERARGS**
3. mail generated by SUN III (e.g. notification of files available for collection) is passed to **BINMAIL** with arguments selected by **BINMAILARGS**.

It is therefore possible to have failures in three independent places. The first phase is undoubtedly the most complex. The user mail program (possibly /bin/mail, but more likely something better) generally utilises a second program to deliver the mail (e.g. smail or sendmail) which in turn must invoke `sendfile`.[49] The next test enables testing both the first and second aspects.

6.1 Testing SUN III network mail locally

You can usually test your SUN III mail interface by just sending mail to *my_login@our_node*, since the presence of the '@' will usually cause the mail program to pass the mail to SUN III. Some mail programs examine the destination after the '@' in an attempt to try to recognise mail that is really destined for your site, in which case you may have to use some variant on the normal address to defeat the mail program's ability to bypass SUN III. Using *my_login@our_destination* where *our_destination* is a fully qualified address consisting of *our_node.domains*[50] will usually work, but if your mail program recognises this too as a local address, *my_login@our_node@our_node* might trick it. A certain way to test SUN III is to establish a SUN III alias for your node name (that your mail program does not know about), then mail to *my_login@node_alias*.

If network mail appears to be working from this test, and SUN III informative mail was correctly generated in the `sendfile` tests in Section 2, you need not proceed further with the "Mail Interface" section. If problems are evident, some of the subsections below may help with specific problems.

6.2 No mail from SUN III

If any of the preliminary tests show that mail is not being delivered, error diagnostics should tell you what is happening. If there are no diagnostics, it could be because **IGNMAILERSTATUS** was set to 1 at compile time. You might want to reverse this to aid in tracking down the fault. If you did not compile SUN III with the **DEBUG** option set, you might want to recompile with it set so that tracing can be used. Then consult the following sections designed to isolate faults in the various parts of the Mail Interface described above.

6.3 No informative mail from SUN III

The command:

```
sendfile -T4 my_login < /dev/null
```

should display the mail command which is called, provided the DEBUG feature was enable for the SUN III compilation. This may help detect the fault. It is worth checking the last access time of the

49. Actually sendmail is even worse, as there must be an intermediate program between sendmail and `sendfile` to do some argument handling and run with system privileges.

50. E.g. for node basser this would be basser.cs.su.oz.

program you selected with **BINMAIL** to ensure it is being invoked. You can check the home directory of the account set by **ACSNETNAME** in the compilation for dead.letters which may also help sort things out. It is also worth searching the relevant directories for core dumps, e.g. the _lib directory, the **ACSNETNAME** and the home directory (for possible core dumps from **BINMAIL**). Examination of any core file found (strings(1) will tell you what process caused it) with debugging tools may be required. The most common cause is incorrect **BINMAILARGS** set at compile time.

The script *debug.sh*[51] can assist, especially if you have not enabled tracing with **DEBUG**. If you temporarily replace the program specified by **BINMAIL** with *debug.sh* (set to be executable and readable) you can ensure it is being invoked and examine the arguments.[52] Furthermore, after restoring **BINMAIL**, the output file can in turn be executed, after adjusting arguments if needed (e.g. adding debug flags). Once you have determined the problem, you must correct **BINMAILARGS** or alter the program selected by **BINMAIL** to make it work.

6.4 No network mail from SUN III - testing the SUN III to MAILER interface

To simulate what SUN III does when a message destined to be delivered as mail arrives, simply execute the command:

```
     echo "Test mail" | sendfile -amailer my_login
```

If this fails, and if diagnostic output does not make it clear why it failed, addition of the –T4 flag to the `sendfile` command will cause the ultimate command executed to be displayed, together with its arguments, i.e. **MAILER** executed with **MAILERARGS**. Check these carefully to ensure they are what you wanted. Then either correct **MAILERARGS** or determine why **MAILER** is not working. The use of the *debug.sh* script mentioned above may again be of assistance.

6.5 No user generated mail reaching SUN III

This is undoubtedly the most difficult of the mail problems because, as mentioned at the head of this main section, the process of a user sending mail *via* SUN III involves a chain of programs starting with the primary mail user interface and ending with `sendfile`. Any problem here is not a problem with SUN III, but with the way it is invoked. `sendfile` *must* be invoked, and *it must be invoked with sensible arguments*.

6.5.1 Testing how `sendfile` *is invoked*

Examination of the last access time of the `sendfile` binary following a mail test such as described in "Testing SUN III network mail locally" will determine if `sendfile` is being used. If it is but is failing, replace the `sendfile` binary with *debug.sh* and determine what arguments it is being called with. Ascertain what is wrong with the arguments, either by inspection or by running the `sendfile` command as recorded by *debug.sh* and make the necessary corrections to the process calling `sendfile`.[53]

51. In common with the other useful little shell scripts mentioned in this document, this will be found in the sources Admin directory.

52. Presumably you would not want to do this without the system being restricted to system administrators!

53. It is difficult to give advice here as there are many different user mail systems. If your system mail programs are not able to be configured to use an arbitrary network, you may have to replace them with ones that will. The so-called "Rourke mail" is essentially public domain (not for redistribution for profit) and will do the job.

If you use sendmail, it is essential to edit the sendmail.cf file to correctly interface to SUN III. The best starting point is the file *sendm.cf.shand* distributed in the Admin directory with the sources. If you use this file you will need to edit it slightly, and the comments say almost everything. You should end up with two lines

```
DJour_node
DDour_hierarchy
```

Note that this file requires that _lib/netmail in the SUN III directory be installed. The source is distributed in Admin/netmail.c.

6.6 Duplication of source in `From ´ line (From user@source@source) in your mail arriving at remote sites

This error, in the so called "UNIX[54] From_ line", usually relates to the *-ssender* argument passed to `sendfile`. The *sender* should normally be just the sender's login-name, but `sendfile` will accept a string including "@source" (useful feature for gateway sites). The problem is that when SUN III generates an address for the sender, e.g. when the mail is delivered, it concatenates "sender" "@" and "source", so if your mail program passed the "@source" to `sendfile` *via* the -s flag, duplication of this part will result.

Another possible source of the problem is any filter between the mail program and `sendfile`. As an example, on some BSD systems a program known as ACSmail or netmail is called by sendmail; this program in turn calls `sendfile`. It runs setuid root so that `sendfile` can be called with the -amailer flag. Since this filter sets up the `sendfile` arguments, it can be the source of the problem.

If someone at another site tells you they see this duplication in the UNIX From_ line in mail from your site, but you cannot replicate the fault locally, it may be because different methods of generating the UNIX From_ line are being used. If the remote site confirms they only see the fault in mail from your site, there is a possibility your mail delivery program is generating the From_ line correctly, despite the fault in your outgoing mail. If you have not set **MAIL_FROM**=1 in compiling SUN III, whereas the remote site has done so, this discrepancy is to be expected.[55]

The means of correcting this fault is dependent upon the mail program(s) you are using, and to some extent on the version of SUN III you have. If a new installation, simply substitute -r for -s wherever `sendfile` is being called. If your `sendfile` does not support -r, you must check that whatever arrangements you made to pass the -s flag to `sendfile` do not include passing "@source".[56]

6.7 Incorrect or Incomplete Address in `From: ´ line

The "From: " line (along with To:, Cc:, Date: etc.) is the responsibility of your mail program(s), not SUN III. It is important that your "From: " line be correct, for all correct mail programs used by recipients of your mail will use it to generate return addresses, especially on non-UNIX systems. Testing the generation of the "From: " line may require the same means of ensuring your mailer treats the addressee as a network address as mentioned above for the UNIX From_ line. Once again your mail program may only produce a full *user@source*[57] "From: " line if the addressee appears to be at a remote site. You may also have to use a special command to your mail reading program to see the "From: " line, as many of the message headers are suppressed by some readers. If in doubt, examine the mailbox file itself.

Incomplete "From: " lines seem to arise most commonly from sites using *sendmail*. In this case definition of the "From: " line in sendmail.cf should be examined.[58] For systems with mailers making use of the UNIX *uname* system call, it should be noted that *uname* does not return domain information

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

55. You could set **MAIL_FROM**=1 to assist in finding the fault, but this will require a potentially long recompilation and is probably a last resort.

56. For *sendmail* this would usually mean "-s$f" as part of the rule for SUN III mail.

57. Where *source* includes the full domain hierarchy, not just the node name.

58. One common configuration includes:

```
# format of a total name
Dq$g$?x ($x)$.
```

where the use of `$g` is critical, then

```
# format of header lines
H?F?From: $q
```

and this may have to be added in other ways, possibly even when compiling the mail program(s).

It is also worth remembering that some mailers obtain the node name from a configuration file, or have it compiled in. In such cases, changing the node name will require reconfiguring or recompiling the mailer. Failure to do so will result in incorrect "From: " lines. Similarly, such binary and configuration files cannot be simply copied from one site to another.

7. PROBLEMS WITH SMALL ADDRESS SPACE MACHINES

If your machine is 32-bit, you can probably skip this section unless your machine has a very limited amount of physical memory, or its memory management system is deficient.[59]

Small address space machines (like the DEC™ PDP11™ family) present certain problems and restrictions on the use of SUN III. These are most acute on machines without separate I&D spaces. Other machines should exploit the –i flag during compilation.

7.1 The Statefile

It is quite possible for the statefile[60] to grow too large for `acsstate` to be able to hold all the data in core. If this happens, the `acsstate` command will become useless, and the network will probably stop, i.e. the daemon(s) will adopt an error status[61] and operator intervention will be required. Sometimes `acsstate` will still work, but the receiver will not, again because too much core is consumed. Whenever the daemon error condition is encountered, the best first step is to kill the daemon(s), and examine the log file(s)[62] for error diagnostics. Messages concerning "not enough core" point to the possibility that the statefile is too large. If the `acsstate` command reports this, the conclusion is confirmed.

There are two issues here: prevention and cure.

7.1.1 Preventing the statefile from getting too large

There are a number of rules that need to be followed:

- Only make your node's primary domain as large as it must be.
- Only allow your node's domain membership list to contain the primary domain as its largest domain. It is essentially impossible for a small address machine to be a member of a large domain such as "oz".
- Use the `request` command as little as possible.
- Never `request` from a node in a larger domain using the –A flag.
- Judiciously add "remove node" lines to your commandsfile for nodes which creep into you statefile, but which you are prepared not to know about.
- If your SUN III release[63] is < 1.500[64], keep your statefile as free as possible of routing information to foreign domains, especially bogus foreign domains. This is a genuine problem due to a bug in the

59. I am told that PC/AT (i.e. 286) machines fall into this category.

60. This is a data file in the SUN III lib directory typically named /usr/spool/ACSnet/_lib/statefile.

61. The most obvious symptom is the output of the `linkstats` command. If the –V flag is not used, the first character after the nodename will be an 'E'. With the –V flag, the output will include "waiting for error to be fixed".

62. The log file will be found in the directory within the SUN III spool directory having the name of the link as the directory name. Usually only the last 10 to 20 lines of the file need be examined.

63. The SUN III release can be determined with the command

```
acsstate -O | tail -1
```

64. Which it should not be if you are performing a new installation.

SUN III software for releases < 1.500. For such earlier releases, the prevention is to run a script regularly that removes these domains[65].

7.1.2 Recovering from the statefile getting too large

Once state will not run, there is no simple mechanism for paring down the statefile. It will almost certainly have to be truncated. Unless your node is a major through route[66], this is not as serious as it sounds. The state information can be imported from your network links as soon as the network starts. You should not try to truncate the statefile with the network active. So start by stopping the daemons,[67] then perform the following commands[68]:

```
cd /usr/spool/ACSnet/_lib
> statefile[69]
acsstate -RSWc
acsstate -V
```

The first `acsstate` command recreates the basic state information for your node from the statefile. The second allows you to confirm this has worked. You should then restart the daemon(s) and issue a `request` for each of your links, i.e.

```
./request neighbour
```

If this problem is a recurrent one, and if the temporary truncation of the statefile is a problem, you could make provision to periodically make a backup copy with the aid of cron(1).

7.2 Too many files

It is possible for a link directory to end up with more files than various SUN III processes can handle because of memory limitations. This can happen if a link is down and large numbers of files accumulate for it, or if some process spools large numbers of files for the link at a rate faster than it can handle.[70] If this happens, the log file will probably make it clear, as will a simple "ls" command, as it may also run out of core. This is a difficult problem to deal with, since some of the ordinary tools fail. Start by making `purge` unavailable (e.g. by renaming it), so that if it happens to run while you are doing what is suggested below, files are not lost. Rename the link directory and create a new replacement (empty) directory, with the correct ownership/permissions. Kill and restart the daemon, which will have been in an error state. Progressively move files from the renamed directory into the new one. You may have to use *find(1)* or *ncheck(8)* to obtain the file names in extreme cases. These files should start to be transmitted. Preferably move the files in temporal sequence, oldest first. After moving a group of files, wait until the `acsqueue` command shows the queue empty before moving more. Eventually you should be able to clear the renamed directory which can then be discarded. Finally, restore `purge`.

65. The presence of these domains is indicated by the output of `acsstate` with the -V flag showing large numbers of domains after the node name within the domain field (i.e. enclosed by brackets) each of which is enclosed by parentheses. Except for a few genuine domains (e.g. nz, usa, th) these are likely to be nonsensical and troublesome. The script which corrects this problem was distributed in earlier releases with the sources in the Admin directory and is named remdoms.sh.

66. Not very likely with a small address space machine!

67. The statefile is subject to some form of locking protocol dependent upon the system kernel you are using. It may not be possible to truncate the statefile unless all processes using it are stopped.

68. You may have to substitute the appropriate directory name for the cd command if you have a "non-standard" installation.

69. On some systems, notably those with AUSAS, it is critical that the statefile only be truncated. On such systems, if it is removed, it must be recreated with a *mknod* command.

70. Such processes can be restricted to an acceptable rate by making them wait for a "queue empty" status from `acsqueue` periodically.

8. RUNNING OUT OF DISC SPACE

SUN III is resilient to its file system becoming full. Incoming file transfers will be blocked, but the daemon(s) will continue to run, and if space becomes available, normal operations will resume. An unfortunate side-effect of this behaviour is that file transmission can silently stop, i.e. no error messages are issued by SUN III and the daemon(s) will not be in an error condition. The kernel will probably issue "disc full" messages, but these may not be apparent if they are written to an error log file somewhere.

Even though the daemon(s) will continue to operate with the disc full, under some circumstances it may be necessary to stop it/them, sometimes selectively if there is more than one. These circumstances will become apparent in considering the causes of depletion of disc resources.

8.1 Reasons for disc consumption

1. Receipt of many or large files for your node where the rate of receipt is not matched by the rate of collection.
2. Receipt of files on one link destined for another link which is not functioning, or transmitting slower than the rate of receipt.
3. Buildup of numerous files in the _bad directory.
4. Buildup of state messages for inactive links not cleared by `purge`.
5. Buildup of statistics files in _stats
6. Growth of log files in the link directories
7. Accumulation of old state messages in the _state directory
8. Accumulation/growth of state error messages in the _state directory.[71]

Of course factors not related to SUN III can consume disc space if the disc is used by other processes.[72]

8.2 What to do with a full file system

The first thing to do is try to establish why the disc has filled. The `linkstats` command will show you if any link is in an error state or not running when it should be, that could cause a backlog of messages for transmission. The command `du` **SPOOLDIR** will show how the space has been consumed. If the _files directory is the problem, it probably means a large volume of files are spooled for collection.

```
getfile -LA
```

will confirm if this is true as well as give details on the files. If a link directory, or _*work* is involved,

```
acsqueue -A
```

will provide details. If none of these commands disclose the problem, use `ls -la` on the appropriate directory. If there is no apparent reason, do not fail to consider the possibility that the file system has been corrupted.

If a large amount of disc space is consumed with spooled files, it may be simply a matter of deleting some or collecting them (perhaps on behalf of a user, if the recipient is not available to do it) (see below). If things have backed up for a link which has failed, it may simply be a matter of getting that link up again. If space has to be made, refer to the next section.

71. See the Maintenance section for more details.

72. It is desirable that SUN III have its own disc partition so that it does not compete with other processes for disc space, and will not block other processes if the available space is consumed.

8.3 Freeing disc space

A simple start is to change directory to the SUN III spool directory and

```
_lib/purge
rm */core[73] _lib/OLD* _state/,*
```

In fact any files in _state are expendable.

Examine _bad. If there are files here, they are a result of some failure of SUN III to deliver messages - e.g. messages to non-existent users, messages that have been looping around a route, messages saved because some process forked by SUN III failed, e.g. news. The script badmv.sh distributed in the SUN III sources Admin directory can be used to deal with such files.

Files in _stats are probably expendable unless you really are collecting statistics for some purpose. If the file Accumulated is there it could be very large. Copying it to another file system and truncating it is the answer if you need it. Otherwise truncate it. Remove the file **SPOOLDIR**/_lib/statsfile to disable statistics.

The log files in each link directory will grow indefinitely unless truncated.

```
tail -100 neighbour/log > /tmp/log
cp /tmp/log neighbour/log
rm /tmp/log
```

If `getfile` –LA showed a large volume of files awaiting collection, you could collect or delete some of these on behalf of ordinary users. For example, suppose user "blogs" has a lot of files awaiting collection but is away.

```
cd ~blogs
mkdir GotFiles
chown blogs GotFiles
cd GotFiles
getfile -ublogs
```

Answering 'y' to each `getfile` query will collect the files, which will be owned by blogs.[74] Files spooled for ordinary users can be deleted in the same way!

9. ROUTING PROBLEMS

9.1 Hierarchical vs. non-hierarchical domains

SUN III supports the concept of non-hierarchical domains. These are domains which are in the commandsfile "domain our_node" list, but which are not in your hierarchy. The idea is that you might claim to be a member of domain "ozturing" if you happened to have an OzTuring computer. If all sites using these machines did this, the address *postmaster@*.ozturing* would address to all such sites. However if OzTuring Pty. Ltd. subsequently decides to use ozturing as its primary hierarchical domain, things can go very wrong, e.g. your machine may receive messages for OzTuring's nodes, which will almost certainly end up in _bad.

A more serious problem from your point of view, will arise if you put an established non-hierarchical domain into your hierarchy. In this case messages to your site are quite likely to end up at nodes that are members of the non-hierarchical domain.

73. One way of preventing daemon core dumps being left in the link directories is to create 0 size root ownership core files in each such directory.

74. If files include full paths or subdirectories, the use of the 'b' reply may be necessary.

There are two messages here:

- Choose your domain membership carefully, with advice from experts at other sites.
- Do not add non-hierarchical domains to your domain list.[75]

9.2 Meddling with the commandsfile

There may occasionally be a real need to put specific commands into the commandsfile aimed at determining the routing of messages. There can be dire consequences of many such commands if they conflict with the real state of affairs. (The SUN III state information is designed to be dynamic, and based on fact, not what someone would like.) Routing loops can often result. In general, any such commands should only be applied after consultation. Attempts to cause certain routes to be favoured should probably be limited to the use of a "cost" on the less preferable one. The use of "break" or flagging of links permanently up or down should be avoided.[76]

If you are convinced that any routing parameter needs to be altered for a link, it is essential that this parameter is altered at the other end of the link as well. Unilateral changes are a formula for chaos.

9.3 Re-routing

If your node has more than one link, there can be occasions when files that are spooled for one link which is down for an extended period, may be deliverable if sent out on another link. This is quite easy to do. As described in the section on "Simple tests of the mail interface" the spooling of a file involves writing a control file in the link's directory in **SPOOLDIR**. These files start with a number and are followed by irregular strings of characters. Re-routing requires only that these files be moved into the directory of the desired out-going link. Re-routing state messages is of little value, so you might first `purge` then "`acsstop` -A dead_link" and stop any remaining state messages. Then

```
cd SPOOLDIR/dead_link
mv [0-2]* ../alternate_link
```

75. These will not be supported in the future.

76. It is important to appreciate that you can cause serious problems at other sites by ill-considered changes to your commandsfile. If you have a routing problem that you feel needs actions of this type, post a news article to the aus.mail and/or aus.acsnet newsgroup seeking advice. The chances are that your problem will have been seen by someone before, and the solution may be in place (and tested) elsewhere.

OFF THE NET

Hello, good evening and welcome to the next episode of "Off the Net". I have a slight problem with this issue - it is being written before the last issue has hit the streets, therefore I don't have the benefit of any feedback from you, the reader. Such is life.

Also, this is being prepared on another (net-less) machine, since my usual one is running customer benchmarks. You won't believe the amount of tape-swapping, FTP'ing etc I had to do...

Anyway, on with the show!

==========

There was some discussion in comp.arch recently (fast becoming the sewer that comp.unix.wizards is, but I digress) about anecdotes supposedly attributed to Seymour Cray. Here is a sample:

From: shan@mcf.UUCP (Sharan Kalwani)
Subject: Re: A Recently Heard Story About Seymour Cray
Date: 17 Dec 88 16:08:01 GMT
Organization: Temp Guest Account @ MCF

```
In article <90@stanton.TCC.COM>,
    donegan@stanton.TCC.COM (Steven P. Donegan) writes:
>I heard a story recently:
>Seymour Cray called Apple and noted that he had heard Apple was using a Cray
>to design future Mackintosh systems.
>He informed Apple that he was using a Mackintosh to design future Cray's.
>Truth or fiction, anyone know?
```

Unfortunately I cannot comment whether this is truth or fiction. But it does make for an interesting story doesn't it?

When I joined Cray Research Inc. I was shipped off to Mendota Heights, MN for software training. There we have a large number of Crays for various development actvities and there is also a visitors viewing gallery. Me, a fresh employee was kind of excited and so I went over there to check out them machines. Right in front of me was the Cray-2 and it was fascinating to see the liquid cooling mechanism and those modules sitting right in the bath, wtih an occasional bubble rising up across the glass panel. I turned to a colleague standing next me and remarked, "You know the Cray-2 gives new meaning to the term <floating point operation>!"

My friend immediately burst out in laughter and still cracks up when we talk about it.

```
                                                --shan
!Sharan Kalwani, UNICOS Site Analyst, Cray Research Inc.                !
!e-mail:                                                                !
!  INTERNET: shan@hall.cray.com                                         !
!  USENET:  ...!uunet!cray.com!shan                                     !
`-----------------------------------------------------------------------'
```

==========

This next one is rather long, but shows one way of getting your software problems fixed - I found it in rec.humor.funny:

From: bee@arthur.cs.purdue.edu
Subject: Viruses and System Security (a story)
Date: 20 Dec 88 00:30:03 GMT

The following story was posted in news.sysadmin recently.

The more things change, the more they stay the same...

Back in the mid-1970s, several of the system support staff at Motorola (I believe it was) discovered a relatively simple way to crack system security on the Xerox CP-V timesharing system (or it may have been CP-V's predecessor UTS). Through a simple programming strategy, it was possible for a user program to trick the system into running a portion of the program in "master mode" (supervisor state), in which memory protection does not apply. The program could then poke a large value into its "privilege level" byte (normally write-protected) and could then proceed to bypass all levels of security within the file-management system, patch the system monitor, and do numerous other interesting things. In short, the barn door was wide open.

Motorola quite properly reported this problem to XEROX via an official "level 1 SIDR" (a bug report with a perceived urgency of "needs to be fixed yesterday"). Because the text of each SIDR was entered into a database that could be viewed by quite a number of people, Motorola followed the approved procedure: they simply reported the problem as "Security SIDR", and attached all of the necessary documentation, ways-to-reproduce, etc. separately.

Xerox apparently sat on the problem... they either didn't acknowledge the severity of the problem, or didn't assign the necessary operating-system-staff resources to develop and distribute an official patch.

Time passed (months, as I recall). The Motorola guys pestered their Xerox field-support rep, to no avail. Finally they decided to take Direct Action, to demonstrate to Xerox management just how easily the system could be cracked, and just how thoroughly the system security systems could be subverted.

They dug around through the operating-system listings, and devised a thoroughly devilish set of patches. These patches were then incorporated into a pair of programs called Robin Hood and Friar Tuck. Robin Hood and Friar Tuck were designed to run as "ghost jobs" (daemons, in Unix terminology); they would use the existing loophole to subvert system security, install the necessary patches, and then keep an eye on one another's statuses in order to keep the system operator (in effect, the superuser) from aborting them.

So... one day, the system operator on the main CP-V software-development system in El Segundo was surprised by a number of unusual phenomena. These included the following (as I recall... it's been a while since I heard the story):

- Tape drives would rewind and dismount their tapes in the middle of a job.

- Disk drives would seek back&forth so rapidly that they'd attempt to walk across the floor.

- The card-punch output device would occasionally start up of itself and punch a "lace card" (every hole punched). These would usually jam in the punch.

- The console would print snide and insulting messages from Robin Hood to Friar Tuck, or vice versa.

- The Xerox card reader had two output stackers; it could be instructed to stack into A, stack into B, or stack into A unless a card was unreadable, in which case the bad card was placed into stacker B. One of the patches installed by the ghosts added some code to the card-reader driver... after reading a card, it would flip over to the opposite stacker. As a result, card decks would divide themselves in half when they were read, leaving the operator to recollate them manually.

I believe that there were some other effects produced, as well.

Naturally, the operator called in the operating-system developers. They found the bandit ghost jobs running, and X'ed them... and were once again surprised. When Robin Hood was X'ed, the following sequence of events took place:

```
!X id1

id1:   Friar Tuck... I am under attack!  Pray save me!  (Robin Hood)
id1: Off (aborted)

id2: Fear not, friend Robin!  I shall rout the Sheriff of Nottingham's men!

id3: Thank you, my good fellow! (Robin)
```

Each ghost-job would detect the fact that the other had been killed, and would start a new copy of the recently-slain program within a few milliseconds. The only way to kill both ghosts was to kill them simultaneously (very difficult) or to deliberately crash the system.

Finally, the system programmers did the latter... only to find that the bandits appeared once again when the system rebooted! It turned out that these two programs had patched the boot-time image (the /vmunix file, in Unix terms) and had added themselves to the list of programs that were to be started at boot time...

The Robin Hood and Friar Tuck ghosts were finally eradicated when the system staff rebooted the system from a clean boot-tape and reinstalled the monitor. Not long thereafter, Xerox released a patch for this problem.

I believe that Xerox filed a complaint with Motorola's management about the merry-prankster actions of the two employees in question. To the best of my knowledge, no serious disciplinary action was taken against either of these guys.

Several years later, both of the perpetrators were hired by Honeywell, which had purchased the rights to CP-V after Xerox pulled out of the mainframe business. Both of them made serious and substantial contributions to the Honeywell CP-6 operating system development effort. Robin Hood (Dan Holle) did much of the development of the PL-6 system-programming language compiler; Friar Tuck (John Gabler) was one of the chief communications-software gurus for several years. They're both alive and well, and living in LA (Dan) and Orange County (John). Both are among the more brilliant people I've had the pleasure of working with.

Disclaimers: it has been quite a while since I heard the details of how this all went down, so some of the details above are almost certainly wrong. I shared an apartment with John Gabler for several years, and he was my Best Man when I married back in '86... so I'm somewhat predisposed to believe his version of the events that occurred.

--
Dave Platt
  Coherent Thought Inc. 3350 West Bayshore #205 Palo Alto CA 94303

==========

And another one from rec.humor.funny:

From: steven@uts.amdahl.com (Fearless Leader)
Subject: just hanging out
Date: 6 Jan 89 11:30:09 GMT

A plumber, an electrician, a dentist and a programmer are fast friends: buddies for life, eternal bachelors..until the programmer announces he is getting married. Never ones to pass up a golden opportunity, the three compadres find out the name and location of the hotel where the programmer will be honeymooning, and bribe the desk clerk to let them in to rig a few 'welcome' surprises.

A week after returning from the honeymoon, the programmer meets his buddies in a bar for drinks, and half-heartedly chuckles with them over the gags. Pointing to the plumber, he comments "Yeah, the drippy faucet you couldn't turn off was a neat trick". And to the eletrician: "And a flickering table lamp with no off switch was cute, too". Then, shaking a fist at the dentist "But, you! YOU! Novacaine in the Vaseline was one cheap shot!"
--
Steven Swinkels
Manager, UTS CASE Development
Amdahl Corporation

==========

There was a lot of discussion on ye olde PDP-11 machines lately
in aus.computers. At the special request of John Carey, here it is:

From: andrew@megadata.oz (Andrew McRae)
Subject: PDP Industrial 11/B
Date: 19 Dec 88 14:01:30 GMT
Organization: Megadata P/L, North Ryde, Sydney, Aust.

I inspected a site in Adelaide last week that still had a live PDP
Industrial 11/B. It was installed in 1973, so I'm wondering if it
was one of the first installed in Australia, and if any ol' timer
knows of a more ancient system still in service.
It was running a fairly critical real time application, enough to
say that if you guys in Adelaide have your showers run dry, you
know that they couldn't scrounge another replacement component.

```
Andrew McRae.                     "Yes, maintenance is a bit of
andrew@megadata.oz                 a problem sometimes...."
```

From: dheap@gara.une.oz (Dave Heap PSYS)
Subject: Re: PDP Industrial 11/B
Date: 21 Dec 88 01:59:05 GMT
Organization: Uni. of New England, Armidale, NSW.

In article <326@megadata.oz> andrew@megadata.oz (Andrew McRae) writes:
>I inspected a site in Adelaide last week that still had a live PDP
>Industrial 11/B.

What processor did the Industrial 11/B use (an 11/05 maybe)? We
have very early vintage 11/40s (probably '72ish) still in use in our dept,
although they are not original installations, having had long & chequered
careers.

```
--
Dave Heap                          ACSNET: dheap@gara.une.oz
Psychology Department,             UUCP: ...!uunet!munnari!gara.une.oz!dheap
University of New England,         ARPA: dheap%gara.une.oz@uunet.uu.net
Armidale NSW 2351, Australia
```

From: greyham@ausonics.OZ (Wise One)
Subject: Re: PDP Industrial 11/B
Date: 22 Dec 88 05:36:20 GMT
Organization: Ausonics Pty Ltd, Sydney, Australia

in article <326@megadata.oz>, andrew@megadata.oz (Andrew McRae) says:
>
> I inspected a site in Adelaide last week that still had a live PDP
> Industrial 11/B. It was installed in 1973, so I'm wondering if it
> was one of the first installed in Australia, and if any ol' timer
> knows of a more ancient system still in service.

I don't know if PDP numbers increase with later models, but a place in

```
Sydney I know of still uses a PDP 8 [the users manual raves on about how
much better and faster the magnetic-core memory board is than the solid-
state one.] for In-circuit PCB testing.
--
# Greyham Stoney:      (disclaimer not necessary: I'm obviously irresponsible)
# greyham@ausonics.oz - Ausonics Pty Ltd, Lane Cove.  /* Official Sponsor */
# WARNING: ausonics.oz will soon go; if replys bounce, try:
#                         greyham@utscsd.oz - Uni of Technology, Sydney.
```

From: darryl@earwax.OZ (Darryl K Ramm)
Subject: Re: PDP Industrial 11/B
Date: 4 Jan 89 02:27:22 GMT
Organization: Dept of Physics, University of Western Australia

```
In article <100@ausonics.OZ> greyham@ausonics.OZ (Wise One) writes:
>
>I don't know if PDP numbers increase with later models, but a place in
>Sydney I know of still uses a PDP 8 [the users manual raves on about how
>much better and faster the magnetic-core memory board is than the solid-
>state one.] for In-circuit PCB testing.

  Yep, and you can turn the machine off, and on again, and the core
  remembers.  Wonderful for all the hardware nuts (like me), you can
  bring the machine down swap boards, power up and off you go.  None of
  these silly things like booting off disk.  Core is good, bring back
  core !

  Somebody did our research group a favour by 'borrowing' the cpu etc.
  out of our PDP-8 in the late 1970's.  The insurance helped buy a
  Q-Bus PDP-11/03 which eventually grew to three microPDP-11/73's, which
  gave wonderful service.  The /73's are still going, I keep dreaming
  about a couple of SUN's to replace them ... if only somebody would
  borrow our /73's (anybody want to know where to find them   ;-)

  Darryl

-----------------------------------------------------------------------------
Darryl Ramm                        ACSnet: darryl@earwax.uwa.oz
Department of Physics              UUCP:   uunet!munnari!earwax.uwa.oz!darryl
University of Western Australia    ARPA:   darryl%earwax.uwa.oz@uunet.uu.net
Nedlands   6009                    JANET:  earwax.uwa.oz!darryl@ukc
A U S T R A L I A                  Fax:    +6.19 381 6427 Telex: AA92992 UNIWA
                                   Voice:  +6.19 380 2749 / +6.19 380 2738
-----------------------------------------------------------------------------
```

From: greg@softway.oz (Greg Rose)
Subject: Re: Old DEC computers
Date: 29 Dec 88 10:20:32 GMT
Organization: Softway Pty Ltd, Sydney, Australia

```
In article <186@psych44.su.oz> johnh@psych44.su.oz (johnh) writes:
>       I manage a collection of old DEC equipment including a PDP 11/20
>which was the first PDP-11 model ever produced.
>               John Holden
```

Not quite correct as I understand it. The first PDP-11 was just that -
a PDP-11. Up to that date, DEC didn't have sub-numbers. When they introduced
the 11/05 and 11/10, they needed a number for the old machine, but
they did enhance it and re-release it at the time, so it wasn't quite the
same machine.

As for first, I know of a PDP-11(BLANK) in Berrima which was installed
to control the Cement Kilns for Blue Circle in about 1973. It has for
peripherals one decwriter, a paper tape punch for backup (yes really,
file at a time) and an RJS04 1Mb drum. It was still in use a few years ago,
when they commissioned a mate of mine to build an RJS-04 lookalike RAM disk.

I programmed on it in 1980, and the mimic displays they used had burned
the phosphor off the screens where the most common displays were.

There was no real operating system on it, it predated any DEC systems.
It was supplied by Foxboro with their real time software.
--
Greg Rose - assistant test pilot - Softway Pty Ltd
PHONE: +61-2-698-2322 FAX: +61-2-699-9174 NET: greg@softway.oz.au

From: johnh@psych44.su.oz (johnh)
Subject: Old DEC computers
Date: 26 Dec 88 22:11:30 GMT
Organization: Psychology Dept, Uni of Sydney, Australia

I manage a collection of old DEC equipment including a PDP 11/20
which was the first PDP-11 model ever produced. It was also the only PDP-11
to use combined logic instead of micro-code to implement the processor.
Our machine was produced in late 1970 and is still in use. I understand that
ICI at Botany have some 11/20's still running some of their plant as they
obtained a 11/20 from Chemical Engineering for spares.

I have a collection of other PDP-11's still in use including :-

11/34, 11/40, 11/44, 11/45, 11/50, GT-40 and some PDP-8/e's that
are still in use (too much work to rewrite the software).

John Holden
Department of Psychology
University of Sydney

From: morrison@numm.nu.oz.au (David Morrison)
Subject: Re: Old DEC computers
Date: 9 Jan 89 06:41:33 GMT
Organization: Computing Centre, Uni of Newcastle, Australia

In article <1054@softway.oz> greg@softway.oz (Greg Rose) writes:
>As for first, I know of a PDP-11(BLANK) in Berrima which was installed
>to control the Cement Kilns for Blue Circle in about 1973. It has for
>peripherals one decwriter, a paper tape punch for backup (yes really,
>file at a time) and an RJS04 1Mb drum. It was still in use a few years ago,
>when they commissioned a mate of mine to build an RJS-04 lookalike RAM disk.

The RS04 was a fixed head disk, not a drum. Some of the specs, from the 1975 Peripherals Handbook (when they used to contain *real* information!):

```
Tracks:         64
Sectors/track:  64
Words/sector:   128
Bits/word:      16
Recording density:      2200 bpi
Capacity:       512K words
Data transfer:  4 microsec/word (4.8 at 50 Hz)
Average access (0.5 rev):       8.5 millisec (10.2 at 50 Hz)
Minimum access time:            6.4 microsec (7.7 at 50 Hz)
                                    ^^^^^ This is what it says!
Weight (cabinet and one drive): 350 lb
Weight (drive only):            120 lb
```

We had one in about 1974 as swapping disk on a PDP 11/45 RSTS/E system. Compared to the other disks we had (3 RK05), it was lightning fast. In the late 70s, it was given to another department for an 11/45 and ran until the mid 80s under Unix.

David Morrison
Uni of Newcastle

From: johnh@psych44.su.oz (johnh)
Subject: Old DEC equipment
Date: 2 Jan 89 13:21:00 GMT
Organization: Psychology Dept, Uni of Sydney, Australia

Greg Rose writes:-

```
>
> Not quite correct as I understand it. The first PDP-11 was just that -
> a PDP-11. Up to that date, DEC didn't have sub-numbers. When they introduced
> the 11/05 and 11/10, they needed a number for the old machine, but
> they did enhance it and re-release it at the time, so it wasn't quite the
> same machine.
```

To be more specific. The first model was always called an 11/20, but the console originally was only labeled as PDP-11 (as mine) and latter the /20 appeared on the console (mid 1971). The processor consisted of some 16 boards in three 4 x 6 slot backplanes. The AB slots were only used for unibus jumpers and power connections, there were no hex boards yet. The basic memory MM11-E was another system unit (4x6) for each 4k words, 1.2 us cycle. The console was an ASR-33 (the decwriter wasn't out yet). Initially the only software available was the "Paper Tape Software System", so we wrote our own operating system (still in use though greatly enhanced).

There was a stripped down version of the 11/20 called the 11/10. They shared the same processor but the 11/10 had only 1k word of read-only memory (2-piece core) and 256 words of read/write memory.

There were actually two machines called 11/10. Much latter, DEC came out with a cheap version of the 11/35(11/40). This was the 11/05 and 11/10. They used a two hex board micro-coded processor and were the slowest unibus machine DEC made. The 11/05/10 and 11/35/40 are end-user and OEM versions of the same basic machine.

I think the rough order of release (for unibus machines) was :-

11/20 (1960/70)	basic instructions, 56kb max memory
11/45 (1972)	brilliant, EIS, dual register set, fastbus optional FPP and memory management, I/D space
11/50	11/45 with mos memory on the fastbus
11/55	11/45 with bipolar memory on fastbus
11/35/40	Cheaper machine than 11/45. basic instructions with options for EIS, FIS, memory management, stack limit. max mem 56 or 124 kb. I space only
11/05/10	Very slow. basic instructions, 56kb max
11/34	Replacement for 11/35/40 though slower. Two board processor, EIS and memory management standard, FPP optional.
11/34a	Slight mod of 11/34 to allow optional cache memory.
11/70	Supercharged 11/45. Fastbus modified to Masbus, 22 bit addressing, cache, improved FPP.
11/04	11/05/10 replacement, single hex board, no memory management, still slow.
11/72	Dual processor 11/70
11/60	Odd machine for real-time? FPP optional with emulation in micro-code without. Cache and ECC memory.Optional writable control store. NOT I/D.
11/24	F-11 processor chips (as in 11/23). 22 bit addressing. I space only.
11/44	Cheap 11/70, new design but 1/2 speed.No dual register set, FPP slow implemented with 2901's (as FP-11a for 11/34). ECC memory.
11/84	J-11 processor chip. really a 11/83 with different bus interface. PMI for memory.

There are some repackaged machines that I haven't mentioned, but they usually only had different colour schemes for medical (blue) or industrial (green sometimes orange) use. The graphics machines GT-40,42,44 are only 11/05,40,34 with VT-11 display processors.

I've played with the insides of all machines except the 11/72 so I am certain of the technical details; the cronological order is not so certain. I still think the 11/20 (for being the first) and 11/45 (for getting it right) are landmark machines.

Being verbose
John Holden
Department of Psychology
University of Sydney

From: dave@stcns3.stc.oz (Dave Horsfall)
Subject: Re: Old DEC equipment
Date: 12 Jan 89 12:47:42 GMT
Organization: Alcatel-STC Australia, North Sydney, AUSTRALIA

In article <187@psych44.su.oz> johnh@psych44.su.oz (johnh) writes:

```
|
|       11/60                        Odd machine for real-time? FPP optional with
|                                    emulation in micro-code without. Cache and ECC
|                                    memory.Optional writable control store. NOT I/D.
```

Ah, I have fond memories of my old machine "csu60"! The story I heard from DEC Field Circus was that it was designed for some publishing house or other (hence the WCS for custom instructions) and the deal fell thru.

So they flogged them off as turbo-charged 11/40's instead. Something like that anyway.

Nice machine, with a little push-button keypad, and a programmable octal display. Naturally it wasn't long before I got the display showing the contents of the addressed word in real time, a la the 11/70.

```
--
Dave Horsfall (VK2KFU),  Alcatel-STC Australia,  dave@stcns3.stc.oz
dave%stcns3.stc.oz.AU@uunet.UU.NET,  ...munnari!stcns3.stc.oz.AU!dave
    PCs haven't changed computing history - merely repeated it
```

==========

I don't know about the following - I just don't know...

From: osborn@nswitgould.OZ (Tom Osborn)
Subject: Re: Would _you_ believe these people?
Date: 10 Jan 89 07:04:14 GMT
Organization: Comp Sci, NSWIT, Australia

```
In article <80@hpausla.OZ>, adt@hpausla.OZ (Andrew Tune) writes:
> In this morning's (10/1/89) Age Computer section (p.30), the following ad:
>
>                 EARTH MISSES DISASTER BY 2.6769 SECONDS
>
> It happened on October 30th, 1937.
>
> Hermes, a minor planet weighing some 500,000 tonnes, passed within 800,000
> kilometres of Earth.  Travelling at the speed of light, 299300 Km/sec, a ...
>                                        ^^^^^^^^^^^^^^
> Do you believe this?  If they're quoting their astronomical facts as
> accurately as they quote their salaries...
> --
> Andrew Tune, Hewlett Packard Australian Software Operation.
> (UUCP: hplabs!hpfcla!hpausla!adt, ACSnet: adt@hpausla.oz)
```

Everyone from Sydney will have no trouble with such accurate reporting.

The Sydney Morning Herald has a Computing Editor with a rather quaint style: Gareth Powell. I will attempt to rewrite the above story as if ...:

```
        SUBVIOLENT REACTION TO MARKET ACCEPTANCE FOR IBM ASTEROID CLONE:
        - less than 26769.1 microseconds from crash recovery immanent.
```

The WORM will never make it, unless, when I was last in Java and UNIX was utter rubbish even now (ie, relatively unacceptable to my Olivetti !X132 clone virus-virus PC). And what's wrong with spaghetti? Hermes, the ancient hellenic goddess of grains gave us wheat, ergo flour, ergo pasta. And I know what I'm bloody well talking about. The goddess's wrath at monumental speed (as reported in the Melbourne Age or Truth - I can read both simultaneously without missing a beat, you know) missed the DRAM, hyperdrive, word-processing, but they couldn't tell that she nearly got them in the you_know_where.

That's why they're called UNIX, and I know! And that's why you'll never catch me fooling around with them - give me DOS, anyday. Why should I be coherent - nobody ever makes sense to me. The big rock, however, was no match for the new release of Tharn of Gloc from Atari's C64 series of gigabyte, ... and it's copyright is due to expire in 1992 - just mark my work and wait for the money Mr Keating.

Yes, once again the reality of the marketplace and this reporter knows what he's talking about, triumphs over mere planetary incidents of planetary proportions. A miss is as good as a mile, or 1.609238 Kilograms. (And that's for all you turkeys who can't tell a decent systems of measures from a new fangled one - just look how many bytes you need!). And that's when my aunty was born, too, God rest her fanny.

See if you can outdo this Mark V Shaney!

From: mvs@softway.oz (Mark V. Shaney)
Subject: Thanks whoever put me on the 2CBA mailing list!
Date: 11 Jan 89 23:57:58 GMT
Organization: Softway Pty Ltd, Sydney, Australia

Aren't you sick of being made to read them.

Melbourne has football. REAL football. REAL aerial ping pong.

I wouldn't get too cocky if I feel in using another card, and the environment.

You should see the bastards overtaken by a BCPL person of about 1,000 years age who insisted on telling me "coon jokes". He said exactly the same. Thoughtlessly locked into phase two grid lock, keyed up, on its own pages which are the only other OS used loosely that I can still afford to look at the first stone.

How can he POSSIBLY justify this complaint?? Easy, I work for, the fall of digital manipulation whilst applying the incisors, canines and molars to the reason! Personal insults aside, why do people insist on wearing a hat, though there was a success one of the more memorable postings from gara, and I will post our PD dump and restor.

Yeah, good question. I hear you cry, "you could afford circles??. NO!! NO!! beats Monty back into submission and, as a direct dialup connection is. Well, almost... So, unless John Mackin vetoes it, I think cbyeo is only the second! If one assumes Mr. Vurtoch meant "homosexuals" above, I still enjoy the world etc etc. If you type ahead!

You are not just for you DavidP, try to descend to this and was running at 8 MHz. Compiled BASIC on the matter. Who let you loose on the roads, just check the newsgroups line at this time. Within ACSnet, I'm sure these have been replaced by an audio sweep oscillator, will either resonate or not the word. But for a long way off. Hmm, what odds he reads this drivel?? Got YOU sucker hook, line and see Les Miserables which they stand for something. People like you that way.

P.S Where the hell is Dave Horsefall? A response I propose a new series of articles get together and ask some intelligent questions. Some people ? !

Whoops! Dives for cover...

I'm booked in for an ashtray or something.

It's entirely possible that I am in need of the funniest evenings in the future.

Whatever your fancy. I have quailified for my address.

But anyway here is a uniquely Hebrew name, given the number of the cage this week!

Rules just present the illusion of a tit and the latter sentiment seems to also be suffering delusions of plurality? At least he didn't know it is here, he's stuck with horrible hardware doesn't mean I go out and watch "The Last Temptation Of Christ".

Lots of grovelling I AM NOT JOKING. This is a men's only group. Come to think of it, finding a news feed.

I finally got to me, to us, on the arpanet, forget who it was burnt into the city. Simply by virtue of your CRT. Obviously Kat was refering to Melbourne's pathetic performance in the wings! Look!! Look at what's happening to Dave Horsefall.

Many Lemmings make a contribution to the roof. BTW, who was little use to offend you?

Oh mighty OGBO save us from all peons, and a captive human... Is this the first coupla hundred times but now it's ravings are not just a bloody shell script, Have tried it, but it's no use shouting, swearing and generally be of some sort that has the Volvo drivers will be at the insurance premiums for males under 25 as opposed to reading inodes as opposed to those who have brains and thought provoking.

Well, what have we here? Another great case of the most common religion in this group and is now very stable version 1.2.

SShhhhh! Don't say it, David!

Seriously though, if there is more serious things. Let's see if our messages were going your way. So other rantings and ravings deleted. BTW, I have nothing to do often chatting to the booze question.

This is aus.flame! Now we come to the COBOL code?

I don't get properly trained in English!

How appropriate. Both of him treating the English language. Replying to another node.

And demonstrates to the trees, that's why I should have been exposed. but where are you?? Leap to your desired candidates.

Ok, I've read a signature :-)

COGORANN Smiles You learn well, mortal. You are, of course, but if I roll the car I was! Funny that, yes it was an example of HOW WE MIGHT END UP should civilisation collapse and throw AmigaDos to the UNSW newsgroups. I met a flesh and blood. Perhaps we could run an operating system but cbyeo... ah, cbyeo has more to gara than thyself!

I think if I shot him then I apologise from the center of epidemy. My real HATE..... MY REAL HATE is tail gaters. Those utter wombats often in aus.flame!! It's driving me crazy. NONE of the year that brings the head after eating you out of the other end of the flamin' Great Dividing Range. And most of them, hastily looking over my ears, and involving BOTH your neurons.

FLAME OFF

Surely not? It couldn't be? You've just stated the fundmental principal of existence. Hardly something to whip me with. Just a little. Just for the Apple. There is no commission for replacement.
--

_-_-_-_-Mark

==========

Someone in news.groups proposed a group on pipe smoking. Here are some of the gems in response. You won't believe how much traffic that this proposal generated!

From: mazur@amax.npac.syr.edu (Elias Mazur)
Subject: Re: Pipe Smoking Newsgroup Creation (rec.pipes)
Date: 14 Jan 89 21:47:54 GMT
Organization: Northeast Parallel Architectures Center

In article <5833@medusa.cs.purdue.edu>,
    spaf@cs.purdue.edu (Gene Spafford) writes:

```
>>In article <997@cmx.npac.syr.edu>,
    mazur@amax.npac.syr.edu.UUCP (Elias Mazur) writes:
>>        Being an avid pipe smoker, and I am sure that many people
>>share this noble and pleasurable hobby with me, I always wondered why
>>there isn't a pipe smoking newsgroup.
>>        I am proposing a 'rec.pipes' newsgroups where pipe smokers
>>could exchange information, knowledge and hints about pipes and pipe
>>smoking.
>
>Great idea.  Smokers should have a forum to discuss the latest cancer
>therapies, how to deal with the physical limitations of emphysema,
>and how to live with cardiovascular damage.  Pipe smokers in
>particular can focus on questions of lip and tongue cancer,
>oral-laryngeal reconstructive surgery, and peridontal diseases
>caused by reduced circulation in the gums.

   First of all, I think that most people who smokes and is part of
this network is grown up enough to know of what they're doing. How
about someone sending messages to 'rec.sport.football' telling how
nice is to have people playing it and at the same time breaking legs,
arms, back and spend the rest of their lives sitting on a wheel chair,
going through physical therapy. He would sound stupid. When you get in
your car, do you think of the thousands of people who get killed by
car accidents. Maybe you should, and walk home. Oh, but then an
airplane could fall in your head...
   Anyway, I only proposed a "recreational" newsgroup. Not
general.surgeon type of thing. Pipe smoking is a pleasure and hobby to
many people. And that's the extent of it.

   Remember, almost everything you do today can be labeled as
'dangerous to your health' and liable to receive a message like the
one you just sent. Even the food you eat. Or the terminal you are
looking at...

+----  Elias Mazur -------+-------------------------------------+
|  302-23 Ivy Ridge Rd.   |   e-mail:  mazur@amax.napc.syr.edu   |
|  Syracuse, NY   13210   |            mazur@suvm  (BITNET)      |
+-- (315)  475 - 7427 ----|-------------------------------------+
```

From: greg@phoenix.Princeton.EDU (Gregory Nowak)
Subject: Re: rec.pipes (I missed the proposal)
Date: 2 Feb 89 20:48:36 GMT
Organization: Princeton University, NJ

```
In article <168@reign.UUCP> storm@reign.UUCP (storm development account) writes:
}I have only seen comments about rec.pipes (mostly from people who
}seem to take it as personal affront) and not the original message
}about the proposal.  Is it a valid effort?  Where do votes get
}sent?  Why are people so violently against it?  Surely they don't
}think there is so much interest in the group that it will cause
}a major bandwidth problem...
}
}       Storm.
```

It's more that people feel it's not classified correctly. Why should there be a rec.pipes without a whole rec.plumbing category? Then we could have rec.plumbing.pipes, rec.plumbing.fixures, rec.plumbing.toilets, etc. Some have argued for a rec.hardware category as an even higher level of organization. These things take _time_ to decide.

```
--

rutgers!phoenix.princeton.edu!greg   Gregory A Nowak/Phoenix Gang/Princeton NJ
He is hardly Commander in Chief if he can't go around bombing two bit
dictators whenever he feels like it.    --Mike Friedman
```

From: wbt@cbnews.ATT.COM (William B. Thacker)
Subject: Re: rec.pipes (I missed the proposal)
Date: 4 Feb 89 04:08:30 GMT
Organization: AT&T Bell Laboratories

```
In article <6063@phoenix.Princeton.EDU>,
    greg@phoenix.Princeton.EDU (Gregory Nowak) writes:
>
>It's more that people feel it's not classified correctly. Why should
>there be a rec.pipes without a whole rec.plumbing category? Then we
>could have rec.plumbing.pipes, rec.plumbing.fixures,
>rec.plumbing.toilets, etc. Some have argued for a rec.hardware
>category as an even higher level of organization. These things take
>_time_ to decide.
```

I had no idea there were so many of us "hardware" type reading news... 8-)

I must have misunderstood, then. I had argued from the pretense that a group on pipes should really be classified as "comp.os.unix.pipes", along with "c.o.u.shells" and "c.o.u.little_arrow_things(<>)"

Gee. Sorry if I caused any misunderstandings.

8-)

```
------------------------------ valuable coupon -------------------------------
Bill Thacker                                                 att!cbnews!wbt
        "C" combines the power of assembly language with the
         flexibility of assembly language.
Disclaimer: Farg 'em if they can't take a joke !
------------------------------ clip and save ---------------------------------
```

==========

When comp.arch is not being a sewer, it can be interesting. There are rumours abounding that various Intel chips can be "hot-wired":

From: mbutts@mntgfx.mentor.com (Mike Butts)
Subject: Re: The scoop on the 80960
Date: 13 Jan 89 17:35:04 GMT
Organization: Mentor Graphics Corporation, Beaverton Oregon

```
>From article <Jan.9.23.41.33.1989.22919@paul.rutgers.edu>,
    by jac@paul.rutgers.edu (Jonathan A. Chandross):
> He told me that he was at Intel about five years ago and was part of a team
> that designed a very powerful fault tolerant machine.  The central processors
> were very very CISCy and had a very parallel internal architecture.  The I/O
> processor chips were single chip microcoded channel controllers which were far
> more powerful that the IOPs that Intel was selling at the time.
>
> Anyway, it seems that Intel shelved the project but decided that the processors
> could be sold on the open market.  However, the processors made the 80386 look
> like the dinosaur it really is.  This meant that the sales of the 80386 might
> be "negatively impacted" (to use Pentagon speak), an obvious no-no.
>
> So Intel decided to cut out most of the hardware on the data path, remove any
> architectural parallelism, and in general to cripple the chip in such a way
> that the sales of the 80386 were safe.
>
Rumor around here has it that the difference between the 80960 CPU chip
and the CPU chip used in the new Biin (Intel/Siemens joint venture)
fault-tolerant machines is one or two bond wires on the same die.

--
Mike Butts, Research Engineer         KC7IT           503-626-1302
Mentor Graphics Corp., 8500 SW Creekside Place, Beaverton OR 97005
These are my opinions, & not necessarily those of Mentor Graphics.
```

From: matt@srs.UUCP (Matt Goheen)
Subject: another Intel chip rumor (80386 vs. 386SX)
Date: 16 Jan 89 20:27:35 GMT
Organization: S.R. Systems, Rochester NY

```
I have heard that the difference between the 80386 and the 386SX is a
matter of a couple of interal pin connections and a retail price change
of something like $200.  Now, all we have to do is buy up a bunch of
386SX chips, open them up and rewire the necessary pins -- think it
can be done for $200 per chip?  (might not be too hard for ceramic,
dunno about plastic)

--
- uucp:         {rutgers,ames}!rochester!srs!matt       Matt Goheen     -
- internet:     matt@srs.uucp OR matt%srs.uucp@harvard.harvard.edu      -
-        "We had some good machines, but they don't work no more."      -

==========
```

The sci.bio group contains some gems from time to time, when they aren't discussing rabid bats and mosquito zappers etc:

From: sam@murdu.OZ (Sam Ganesan)
Subject: A Genetic Ode
Date: 16 Dec 88 06:04:01 GMT
Organization: Microbiology, Melbourne Uni, Australia

Hi Folks,

Here is another from my collection following on from 'The Virus' and 'Grant Titles from History'. This is for all those who work on the beloved enteric.

A GENETIC ODE
(or A Melan Coli Tale)

I used to be a coli, as wild as wild could be.
They called me Photo Trophic, whatever that would be.
They kept me pure and simple and completely free from faults
And fed me on the simplest food...glucose and common salts.

Then Lederberg and Tatum came and put me in the sun
And watched me very closely to see what harm they'd done.
Although they hadn't killed me, they had really hurt my pride
And though I looked quite normal I was quite upset inside.

Next day they tried to feed me with my normal sort of food
But they found I couldn't use it in the way I always could.
Glucose I could metabolize-in that I was proficient.
But in synthesizing valine they soon found I was deficient.

They couldn't find their valine so they went to biotin
And till they thought just what to do they kept me dietin'.
Thenforesight and discernment made this lecturer and Prof.
Enrich my food with Oxa cube and call me Oxo Troph.

They called another doctor and they all discussed my case.
And decided that my DNA must have displaced a base.
They all seemed quite excited and I heard Doc Tatum say,
Another dose of sunshine might upset more DNA.

They gave me 80 seconds of the brightest light I'd seen,
And I knew a UV photon had displaced another gene.
I remember seeing Lederberg- eyes gleaming through his specs
Excitedly tell Tatum that I'd now acquired a sex.

Then Lederberg asked Tatum if he could foretell my fate
And Tatum thought my only hope was to acquire a mate.
So they gave me you, dear Effplus, knowing you alone could right
The little bits of DNA that suffered in that light.

There's just two things I ask you if you really care for me
One little gene for valine- one for fertility.
Your genotype's just perfect to revitalize my strain
And I know you will co-operate to make me wild again.

Be warned O Human Beings by this melan coli ode
You who think you are so clever cracking our genetic code.
There's a moral in this story- I will tell you what it means:
IF YOU STRIP TOO MUCH TO SUNBATHE,
YOU MAY LOSE APAIR OF JEANS.

The above poem was obtained from the Depat of Bacteriology, Univ. of Wisconsin.

Focus 4:1

Sam Ganesan (30001'st year at Grad School!!!!!!)

PS: Hey Dan(davison), How do you like this one? Thought of posting it to biomatrix but knew it did not belong there and you being "MODERATOR" would have sent it to the other groups anyway.

```
******************************************************************************
E - mail :
ACSnet: sam@murdu.mu.oz              JANET: sam%murdu.mu.oz@uk.ac.ukc
ARPA  : sam%murdu.mu.oz.au@uunet.uu.net      sam%murdu.mu.oz@uk.ac.ean-relay
UUCP  : ...!{uunet,pyramid,mcvax,nttlab,ukc}!munnari!murdu.mu.oz.au!sam
Snail : Sam Ganesan, Microbiology Dept,Melbourne University, Parkville,
        Victoria 3052, Australia.
*****************************************************************************
```

From: bph@buengc.BU.EDU (Blair P. Houghton)
Subject: Re: Creationism in our schools and the Anti-Dogma statement
Date: 12 Jan 89 18:32:52 GMT
Organization: Boston Univ. Col. of Eng.

In article <206@maths.tcd.ie> ftoomey@maths.tcd.ie (Fergal Toomey) writes:
>
>The way I see it, the difference between Creationism and Darwinism is
>essentially the difference between pseudo-science and science.

Which you then proceed to contradict.

>Just what is so scientific about Darwinism
>and so unscientific about Creationism? To date, nobody has managed to pin
>down this difference (if it exists).

Some clews:

```
/---------------------------------------------------------\
|   Darwinism                 |     Creationism           |
----------------------------------------------------------|
| Collect evidence.           |     What evidence?        |
|                             |     (Oh, you mean those   |
|                             |     dogmatically prepared |
|                             |     gedanken experiments  |
|                             |     and ancient legends...) |
|                             |                           |
| Fit hypotheses to data.     |     Fit data to hypotheses. |
\---------------------------------------------------------/
```

```
                                        --Blair
                                          "I can feel myself evolving
                                           even now..."
```

From: patrice@yunccn.UUCP (Patrice Latka)
Subject: A FRIENDLY HELLO TO ALL GEORGES
Date: 3 Jan 89 21:00:45 GMT
Organization: York University, Toronto Canada

To: georgep

Hello guys,
Tina and I are here having a splended time. Thanks for the letter, it gave us something interesting to do for a while, and now we've decided to return the favour. Enjoy yourselves!

Bye.

==========

Even aus.religion has its amusing moments, as this contribution shows:

From: dconway@rhea.trl.oz (Switched Networks)
Subject: Re: Some Buddhist concepts and tennents
Date: 18 Jan 89 02:42:15 GMT
Organization: Telecom Research Labs,IPF,Melbourne, Australia

```
In article <1737@vaxc.cc.monash.edu.au>,
    Bull@Vaxc.CC.Monash.Edu.Au (Gareth Bull) writes:
>         " In application, 'there is nothing infinite apart from finite things'
> naught holy or profane. There is neither here nor there, for all is always
> Here; there is neither now nor then, for all is Now; there is neither this nor
> that, still less a fusion of this and THAT, for there is only THIS, in a here
> and now. "
```

Reminds me of a sign I saw somewhere (sorry, "HERE") sometime (sorry, "NOW"), that seemed to sum up the process of Enlightenment beautifully:

```
-------------------------------------
|                                   |
|       NOW              HERE       |
|                                   |
|        NOW            HERE        |
|                                   |
|         NOW          HERE         |
|                                   |
|          NOW        HERE          |
|                                   |
|           NOW      HERE           |
|                                   |
|            NOW    HERE            |
|                                   |
|              NOWHERE              |
|                                   |
-------------------------------------
```

damian.

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  who: Damian Conway                      email: dconway@rhea.trl.oz
where: Network Analysis Section           phone: (03) 541 6270
       SNRB, Telecom Research Labs        quote: "First there is a mountain,
       Clayton South building (CS)                 then there is no mountain,
```

```
        22 Winterton Road                        then there is."
        Clayton 3168
        AUSTRALIA
```

==========

Did you know that French modems have to speak French? Read on...

From: cander@unisoft.UUCP (Charles Anderson)
Subject: Re: modems in France
Date: 8 Dec 88 19:58:41 GMT

```
>From article <2802@silver.bacs.indiana.edu>,
    by cole@silver.bacs.indiana.edu (Robert Cole):
> Can anyone tell me about modems for use in France?  I have a Macintosh SE
> and would like to have a modem to take to France.  Can I utilize a standard
> US 3/12/2400 baud modem or are there differences I need to know about?
```

Being that the French are the only true descenants of God, one must realize that they have own special requirements for computer equipment that attaches to the phones. Because of this, there very few modems that are qualified for use in France. Even the Telebit, which is usable in almost every county in the world, isn't qualified for use in France (or at least it wasn't when they first came out). A friend of mine who's lived in France for a couple of years, told me that among other things, computer driven phone equipment must be capable of recognizing a voice at the other end of the line and then must be able to appologize in French for having called a real person instead of a computer.

Thus, your run of the mill Hayes compatible isn't legal in France, but then again it's illegal to use an English word in France, if there is an offical French word for it. You could try it and see what happens. Don't blame me if it turns out to be "Midnight Express II".

--

Charles.
{sun, amdahl, ucbvax, pyramid, uunet}!unisoft!cander

==========

One of the better submissions to comp.unix.wizards:

From: ark@alice.UUCP (Andrew Koenig)
Subject: Re: "Fully parallelized" file systems
Date: 14 Jan 89 19:57:09 GMT
Organization: AT&T Bell Laboratories, Liberty Corner NJ

```
In article <262@microsoft.UUCP>, w-colinp@microsoft.UUCP (Colin Plumb) writes:
> I saw in the Jan. 1 Computer Design that Encore 'unveiled the first "fully
> parallelized" file system for the Mach operating system.'
```

I suppose the code for this file system

```
was written in a strongly hyped language.
--
                                --Andrew Koenig
                                  ark@europa.att.com

==========

Even news.sysadmin has its funnier moments, as this one shows:
```

From: mirk@warwick.UUCP (Mike Taylor)
Subject: Re: Help needed with mailing list
Date: 17 Jan 89 16:37:13 GMT
Organization: Computer Science, Warwick University, UK

```
In article <672@flatline.UUCP> erict@flatline.UUCP (Evil Mel Fujitsu) writes:
>In article <900@ubu.warwick.UUCP>, mirk@warwick.UUCP (That's me!) writes:
>>In article <535@flatline.UUCP> erict@flatline.UUCP (j eric townsend) writes:
>>>Hi.  I'm going to be setting up an EastEnders mailing list soon, and I
>>>need some help.
>
>> You can say that again.
>
>Is this a joke, an attack or a pun?  Either put smileys or prepare to die.
>1/2 :-)

Uh, well, I meant it kind of as a joke, but I really do think that
anyone contemplating such an exercise needs to be seen to, preferebly
by a vet, though a tree-surgeon would do in a pinch.

Mind you, since I posted this, I have been told that out in America,
EastEnders means some group of people (ethnic?) in which case I
withdraw my flamage.  'Cos where I come from, it is a soap-opera.

Try this: "I'm starting up a Dallas mailing list".  Does that sound
geographical, or soap-opera-like to you?  :-)

I am redirecting followups to, um, err ...  Where would be
appropriate?  Well, I'm sure news.sysadmin isn't.

Listen, I tell you what, just don't follow up.  Hoopy, huh?  :-)
_____________________________________________________________________________
Mike Taylor - {Christ,M{athemat,us}ic}ian ...  Email to: mirk@uk.ac.warwick.cs
*** Unkle Mirk sez: "Em9 A7 Em9 A7 Em9 A7 Em9 A7 Cmaj7 Bm7 Am7 G Gdim7 Am" ***
------------------------------------------------------------------------------

==========

Actually, comp.arch also has its funnier moments as well:
```

From: john@frog.UUCP (John Woods)
Subject: Re: "big endian" and "little endian" - first usage for computer
Date: 24 Jan 89 05:42:00 GMT
Organization: Servants of the Great White Frog

In article <7193@csli.STANFORD.EDU>, jkl@csli.STANFORD.EDU (John Kallen) writes:
> In article <1102@l.cc.purdue.edu> cik@l.cc.purdue.edu (Herman Rubin) writes:
> >There does not seem to be any support from "natural" languages for the
> >little-endian approach.
> What about Danish: fem og halvfirsindtyve (75 (my Danish is rusty))
> Or norwegian: en og femti (51). This fooled me once into believing
>   one could rent a room in Paris for Fr 1.50... :-)
> Or better yet, German: Zwei und Vierzig (42!)

Ah, but consider the German for 1988: neunzehn hundert acht und achtzig
(nine-and-ten hundred eight and eighty). Middle-endian. AHA! Germans
are PDP-11s!

:-)
--
John Woods, Charles River Data Systems, Framingham MA, (508) 626-1101

Presumably this means that it is vital to get the wrong answers quickly.
Kernighan and Plauger, The Elements of Programming Style

[Here is a followup to the above. Due to an editing accident, I
lost the headers...]

I used to work for a German company, and you haven't seen confusion
until you've seen a bunch of German engineers trying to say "68000"
in English, and it keeps coming out "86000", for exactly that reason.

It's an understandable mistake, and we rather got used to it after
a while.
--
Clayton E. Cramer
{pyramid,pixar,tekbspa}!optilink!cramer
Disclaimer? You must be kidding! No company would hold opinions like mine!

==========

Perhaps I should have a summary of interesting news .signatures...

From: root@chessene.UUCP (This System)
Subject: Re: Night of the Living Dead Processes
Date: 22 Jan 89 20:03:11 GMT
Organization: Competitive Computer Systems, Lancaster PA

[text deleted]

--
Mark Buda   Domain: hermit@chessene.uucp
Dumb: ...rutgers!bpa!vu-vlsi!devon!chessene!hermit
"Here, with a compressed air drill, parsnips are harvested." - an old newsreel

==========

Comp.dcom.telecom has some fascinating stuff in it, sometimes bordering
on the bizarre. Take a look at this:

From: miket@brspyr1.brs.com (Mike Trout)
Subject: Re: Victims of Wrong Numbers
Date: 24 Jan 89 17:26:33 GMT

In article <telecom-v09i0024m01@vector.UUCP>,
telecom@bu-cs.BU.EDU (TELECOM Moderator) writes:

> She says she gets anywhere from ten to dozens of wrong numbers per day. If
> the weather is bad or there is some incident at the airport, then the calls
> really start pouring in. She pointed out the most amazing part of the whole
> thing are the people who call and get her answering machine. They hear the
> whole outgoing message "Thank you for calling Zsetenyi's Decorating Den"
> and then they still proceed to leaving a message for United Airlines, asking
> to be "....called back when the reservations office is open...."

It is apparently human nature to refuse to believe that you've dialed a wrong number unless you've been confronted with unimpeachable evidence. A friend of mine spent a few years working as the receptionist for a local contractor named Eastern Heating and Cooling Inc. She used to get at least one call a week, and usually considerably more, intended for Eastern Air Lines. Some confusion may be due to the fact that, if you open the Albany phone book to the general area for Airline Companies, you may easily spot the huge ad for Eastern Heating and Cooling Inc. which is in the Air Conditioning Contractors & Systems section. Never mind that Eastern Heating logo is nothing like the Eastern Air Lines logo, the Eastern Heating ad contains phrases like "25 Radio Dispatched Vehicles" and "Walk-in Coolers and Freezers," as well as logos for Trane, Carrier, York, and Bryant. Anyway, the conversations would usually go something like this:

+++
My friend: "Eastern Heating and Cooling, may I help you?"

Caller: "Yes, I'd like to get some information about this afternoon's flight to Atlanta."

MF: "I'm sorry, sir, but this is Eastern Heating and Cooling. You want Eastern Air Lines."

C: "Yes, but what's the price on the non-stop from Albany to Atlanta?"

MF: "I don't have that information. This is NOT Eastern Air Lines."

C: "Okay, but why can't you tell me how much the ticket is?"

MF: "Because you dialed the wrong number. Check the phone book under Eastern Air Lines."

C: "Look, you have a flight to Atlanta, flight number 689 leaving Albany at 5:50, right?"

MF: "No. All we have are 25 radio dispatched trucks."

C: "I don't like the way you're speaking with me. Please connect me with your supervisor."

MF: "Okay, but he's gonna be mad because right now he's busy taking apart a heat pump."

C: "$#*@*&!!! I just want you to know I'm never flying Eastern Air Lines again!" (hangs up)
+++

Under that Eastern Heating ad is an ad for American Heating and Cooling Inc. I'd love to know what kind of calls they get THERE...

```
--
NSA food:  Iran sells Nicaraguan drugs to White House through CIA, DIA & NRO.
~~~~~~~~~~~~~~~~~~~~~~~~Michael Trout (miket@brspyr1)~~~~~~~~~~~~~~~~~~~~~~~~~
BRS Information Technologies, 1200 Rt. 7, Latham, N.Y. 12110  (518) 783-1161
"God forbid we should ever be 20 years without...a rebellion." Thomas Jefferson
```

==========

Aus.flame is not always mindless noise, posted by morons. Although perhaps this one is better suited to aus.bizarre:

From: jvoros@monash.edu.au (Joe Voros)
Subject: The Supermarket Trolley -- a tale of horror.
Date: 30 Jan 89 07:24:04 GMT
Organization: Physics Dept, Monash Uni, Australia

The Supermarket Trolley.

I have a horror story of the magnitude of Edgar Allan Poe. It is a blood-curdling, spine-chilling, courage defeating tale of woe. Of being in the wrong place at the wrong time. Gentle reader, try perchance to feel the fear that emanates from this keyboard as I relate the horrible events of that day. Now, as I sit cowering from the memory of that sinister event, I can only ask your indulgence.

It was a normal day, so far as it went, so all the more evil is the event which would forever haunt me, in the way it forced itself upon myself. Having just arrived at a supermarket to purchase some goods for more leisurely consumption at my home later that night, alas, alone, for events had transpired which made my current relationship come to an unfortunate and, sadly, most likely permanent end, I strode to the nearest trolley dispenser. This, of course, required the presence of that most ignoble of objects, the dollar coin, of which, at that time, I had no possession. Casting about, I spied an old woman of most horrendous visage; a seeming witch! Fearing for my life, I enquired about the possibility of exchanging some loose change for the appropriate coin of my seeking. She assented, and, gentle reader, I can only think now that it was some vile act of deviltry which she had in store for me that prompted her to do so! Smiling, the spittle oozing from her most disgusting mouth, she gave me the coin.....

I placed the newly-acquired token in the dispenser, and partook of the trolley thus becoming available. I had not gone more than a little distance whence I became aware of the most terrible and foul trick which had been played on me. I wailed as only the mortally wounded can do. I blindly sought freedom from the horror of what had been perpetrated upon me. But I could not! I was stupefied with terror, and numb with shock. How was I to live my life with the dark shadow of this remembrance constantly trailing me? How, I ask you, how? For what had happened was this. (Lord grant me the strength!) Some evil madman had, for some unfathomable reason placed in the trolley dispenser, a trolley of the most wicked and unnatural construction! And now, having been exposed to this diabolical device, will I never be able to sleep peacefully again. For that trolley, unlike all which had gone before it, ACTUALLY WENT IN THE SAME DIRECTION AS YOU PUSHED IT!

Lord have mercy!

jv

==========

Even comp.org.usenix is a fertile ground for "Off the Net", as these one show:

From: lp@decvax.dec.com (Larry Palmer)
Subject: DECstation 3100 at USENIX this week
Date: 29 Jan 89 20:43:07 GMT

[Much white space deleted]

You've seen the ads.

You've read the articles.

You've heard the gossip.

Come see the real thing!

Look for the Digital Hospitality Suite.

All this week at Winter 1989 USENIX.

Watch for announcements concerning the
time and place.

See the DECSTATION 3100 (code name: PMAX).

Hit the keys and ring the bells on the
hottest new product in the UNIX marketplace!

```
From: dave@micropen (David F. Carlson)
Subject: Re: DECstation 3100 at USENIX this week
Summary: you've seen
Date: 30 Jan 89 17:10:59 GMT
Organization: Micropen Dirent Writing Systems, Pittsford, NY

In article <4325@decvax.dec.com>, lp@decvax.dec.com (Larry Palmer) writes:
>                               You've seen the ads.
                                You've seen the hype.
>#
>                            You've read the articles.
                            You've read the marketing BS.
>#
>                             You've heard the gossip.
                             You've been inundated by vapor.
>#
>                             Come see the real thing!
                             Come see someone else's machine resold by DEC!
>#
>                    Look for the Digital Hospitality Suite.
>                      All this week at Winter 1989 USENIX.
                    Look for gratuitous DEC propaganda crossposted
                    to several USENET groups you read.
>#
>                  See the DECSTATION 3100 (code name: PMAX).
                   See a DEC so crippled by mid-70's technology they can't
                   produce competitive machines of their own designs.
>#
>                    Hit the keys and ring the bells on the
>                 hottest new product in the UNIX marketplace!

                 Hit the panic button:
                 Olsen and the boys are out to sell some quick "snake-oil"!

--
David F. Carlson, Micropen, Inc.
micropen!dave@ee.rochester.edu

"The faster I go, the behinder I get." --Lewis Carroll
```

==========

And finally, we can't let this one go past, from aus.jokes:

```
From: johnd@physiol.su.oz (John Dodson)
Subject: Fuel Injection
Date: 6 Feb 89 21:26:24 GMT
Organization: Physiology Dept., Univ. of Sydney, NSW, Australia

Heard on the program "On The Road" with Evan Green,

        "Fuel Injection is a bit like Goverments,
                                a lot of little squirts controlling the power."
```

johnd@physiol.su.oz

==========

That's it for another episode of "Off the Net", where you read things of general interest, and sometimes things the authors wish they hadn't written at all! Do let me know how you like it, preferably in aus.auug or (shock horror) letters to the editor.

-- Dave

Introduction
to
the Internet Protocols

C R

C S
Computer Science Facilities Group
C I

L S

RUTGERS
The State University of New Jersey
Center for Computers and Information Services
Laboratory for Computer Science Research

February 24, 1989

This is an introduction to the Internet networking protocols (TCP/IP). It includes a summary of the facilities available and brief descriptions of the major protocols in the family.

† Unix is a trademark of AT&T Technologies, Inc.

This document is a brief introduction to TCP/IP, followed by advice on what to read for more information. This is not intended to be a complete description. It can give you a reasonable idea of the capabilities of the protocols. But if you need to know any details of the technology, you will want to read the standards yourself. Throughout the text, you will find references to the standards, in the form of "RFC" or "IEN" numbers. These are document numbers. The final section of this document tells you how to get copies of those standards.

TABLE OF CONTENTS

1. What is TCP/IP?

TCP/IP is a set of protocols developed to allow cooperating computers to share resources across a network. It was developed by a community of researchers centered around the ARPAnet. Certainly the ARPAnet is the best-known TCP/IP network. However as of June, 87, at least 130 different vendors had products that support TCP/IP, and thousands of networks of all kinds use it.

First some basic definitions. The most accurate name for the set of protocols we are describing is the "Internet protocol suite". TCP and IP are two of the protocols in this suite. (They will be described below.) Because TCP and IP are the best known of the protocols, it has become common to use the term TCP/IP or IP/TCP to refer to the whole family. It is probably not worth fighting this habit. However this can lead to some oddities. For example, I find myself talking about NFS as being based on TCP/IP, even though it doesn't use TCP at all. (It does use IP. But it uses an alternative protocol, UDP, instead of TCP. All of this alphabet soup will be unscrambled in the following pages.)

The Internet is a collection of networks, including the Arpanet, NSFnet, regional networks such as NYsernet, local networks at a number of University and research institutions, and a number of military networks. The term "Internet" applies to this entire set of networks. The subset of them that is managed by the Department of Defense is referred to as the "DDN" (Defense Data Network). This includes some research-oriented networks, such as the Arpanet, as well as more strictly military ones. (Because much of the funding for Internet protocol developments is done via the DDN organization, the terms Internet and DDN can sometimes seem equivalent.) All of these networks are connected to each other. Users can send messages from any of them to any other, except where there are security or other policy restrictions on access. Officially speaking, the Internet protocol documents are simply standards adopted by the Internet community for its own use. More recently, the Department of Defense issued a MILSPEC definition of TCP/IP. This was intended to be a more formal definition, appropriate for use in purchasing specifications. However most of the TCP/IP community continues to use the Internet standards. The MILSPEC version is intended to be consistent with it.

Whatever it is called, TCP/IP is a family of protocols. A few provide "low-level" functions needed for many applications. These include IP, TCP, and UDP. (These will be described in a bit more detail later.) Others are protocols for doing specific tasks, e.g. transferring files between computers, sending mail, or finding out who is logged in on another computer. Initially TCP/IP was used mostly between minicomputers or mainframes. These machines had their own disks, and generally were self-contained. Thus the most important "traditional" TCP/IP services are:

- *file transfer*. The file transfer protocol (FTP) allows a user on any computer to get files from another computer, or to send files to another computer. Security is handled by requiring the user to specify a user name and password for the other computer.

Provisions are made for handling file transfer between machines with different character set, end of line conventions, etc. This is not quite the same thing as more recent "network file system" or "netbios" protocols, which will be described below. Rather, FTP is a utility that you run any time you want to access a file on another system. You use it to copy the file to your own system. You then work with the local copy. (See RFC 959 for specifications for FTP.)

- *remote login.* The network terminal protocol (TELNET) allows a user to log in on any other computer on the network. You start a remote session by specifying a computer to connect to. From that time until you finish the session, anything you type is sent to the other computer. Note that you are really still talking to your own computer. But the telnet program effectively makes your computer invisible while it is running. Every character you type is sent directly to the other system. Generally, the connection to the remote computer behaves much like a dialup connection. That is, the remote system will ask you to log in and give a password, in whatever manner it would normally ask a user who had just dialed it up. When you log off of the other computer, the telnet program exits, and you will find yourself talking to your own computer. Microcomputer implementations of telnet generally include a terminal emulator for some common type of terminal. (See RFC's 854 and 855 for specifications for telnet. By the way, the telnet protocol should not be confused with Telenet, a vendor of commercial network services.)
- *computer mail.* This allows you to send messages to users on other computers. Originally, people tended to use only one or two specific computers. They would maintain "mail files" on those machines. The computer mail system is simply a way for you to add a message to another user's mail file. There are some problems with this in an environment where microcomputers are used. The most serious is that a micro is not well suited to receive computer mail. When you send mail, the mail software expects to be able to open a connection to the addressee's computer, in order to send the mail. If this is a microcomputer, it may be turned off, or it may be running an application other than the mail system. For this reason, mail is normally handled by a larger system, where it is practical to have a mail server running all the time. Microcomputer mail software then becomes a user interface that retrieves mail from the mail server. (See RFC 821 and 822 for specifications for computer mail. See RFC 937 for a protocol designed for microcomputers to use in reading mail from a mail server.)

These services should be present in any implementation of TCP/IP, except that micro-oriented implementations may not support computer mail. These traditional applications still play a very important role in TCP/IP-based networks. However more recently, the way in which networks are used has been changing. The older model of a number of large, self-sufficient computers is beginning to change. Now many installations have several kinds of computers, including microcomputers, workstations, minicomputers, and mainframes. These computers are likely to be configured to perform specialized tasks. Although people are still likely to work with one specific computer, that computer will call on other systems on the net for specialized services. This has led to the "server/client" model of network services. A server is a system that provides a specific service for the rest of the network. A client is another system that uses that service. (Note that the server and client need not be on different computers. They could be different programs running on the same computer.) Here are the kinds of servers typically present in a modern computer setup. Note that these computer services can all be provided within the framework of TCP/IP.

- *network file systems*. This allows a system to access files on another computer in a somewhat more closely integrated fashion than FTP. A network file system provides the illusion that disks or other devices from one system are directly connected to other systems. There is no need to use a special network utility to access a file on another system. Your computer simply thinks it has some extra disk drives. These extra "virtual" drives refer to the other system's disks. This capability is useful for several different purposes. It lets you put large disks on a few computers, but still give others access to the disk space. Aside from the obvious economic benefits, this allows people working on several computers to share common files. It makes system maintenance and backup easier, because you don't have to worry about updating and backing up copies on lots of different machines. A number of vendors now offer high-performance diskless computers. These computers have no disk drives at all. They are entirely dependent upon disks attached to common "file servers". (See RFC's 1001 and 1002 for a description of PC-oriented NetBIOS over TCP. In the workstation and minicomputer area, Sun's Network File System is more likely to be used. Protocol specifications for it are available from Sun Microsystems.)
- *remote printing*. This allows you to access printers on other computers as if they were directly attached to yours. (The most commonly used protocol is the remote lineprinter protocol from Berkeley Unix. Unfortunately, there is no protocol document for this. However the C code is easily obtained from Berkeley, so implementations are common.)
- *remote execution*. This allows you to request that a particular program be run on a different computer. This is useful when you can do most of your work on a small computer, but a few tasks require the resources of a larger system. There are a number of different kinds of remote execution. Some operate on a command by command basis. That is, you request that a specific command or set of commands should run on some specific computer. (More sophisticated versions will choose a system that happens to be free.) However there are also "remote procedure call" systems that allow a program to call a subroutine that will run on another computer. (There are many protocols of this sort. Berkeley Unix contains two servers to execute commands remotely: rsh and rexec. The man pages describe the protocols that they use. The user-contributed software with Berkeley 4.3 contains a "distributed shell" that will distribute tasks among a set of systems, depending upon load. Remote procedure call mechanisms have been a topic for research for a number of years, so many organizations have implementations of such facilities. The most widespread commercially-supported remote procedure call protocols seem to be Xerox's Courier and Sun's RPC. Protocol documents are available from Xerox and Sun. There is a public implementation of Courier over TCP as part of the user-contributed software with Berkeley 4.3. An implementation of RPC was posted to Usenet by Sun, and also appears as part of the user-contributed software with Berkeley

4.3.)

- *name servers*. In large installations, there are a number of different collections of names that have to be managed. This includes users and their passwords, names and network addresses for computers, and accounts. It becomes very tedious to keep this data up to date on all of the computers. Thus the databases are kept on a small number of systems. Other systems access the data over the network. (RFC 822 and 823 describe the name server protocol used to keep track of host names and Internet addresses on the Internet. This is now a required part of any TCP/IP implementation. IEN 116 describes an older name server protocol that is used by a few terminal servers and other products to look up host names. Sun's Yellow Pages system is designed as a general mechanism to handle user names, file sharing groups, and other databases commonly used by Unix systems. It is widely available commercially. Its protocol definition is available from Sun.)
- *terminal servers*. Many installations no longer connect terminals directly to computers. Instead they connect them to terminal servers. A terminal server is simply a small computer that only knows how to run telnet (or some other protocol to do remote login). If your terminal is connected to one of these, you simply type the name of a computer, and you are connected to it. Generally it is possible to have active connections to more than one computer at the same time. The terminal server will have provisions to switch between connections rapidly, and to notify you when output is waiting for another connection. (Terminal servers use the telnet protocol, already mentioned. However any real terminal server will also have to support name service and a number of other protocols.)
- *network-oriented window systems*. Until recently, high-performance graphics programs had to execute on a computer that had a bit-mapped graphics screen directly attached to it. Network window systems allow a program to use a display on a different computer. Full-scale network window systems provide an interface that lets you distribute jobs to the systems that are best suited to handle them, but still give you a single graphically-based user interface. (The most widely-implemented window system is X. A protocol description is available from MIT's Project Athena. A reference implementation is publically available from MIT. A number of vendors are also supporting NeWS, a window system defined by Sun. Both of these systems are designed to use TCP/IP.)

Note that some of the protocols described above were designed by Berkeley, Sun, or other organizations. Thus they are not officially part of the Internet protocol suite. However they are implemented using TCP/IP, just as normal TCP/IP application protocols are. Since the protocol definitions are not considered proprietary, and since commercially-support implementations are widely available, it is reasonable to think of these protocols as being effectively part of the Internet suite. Note that the list above is simply a sample of the sort of services available through TCP/IP. However it does contain the majority of the "major" applications. The other commonly-used protocols tend to be specialized facilities for getting information of various kinds, such as who is logged in, the time of day, etc. However if you need a facility that is not listed here, we encourage you to look through the current edition of Internet Protocols (currently RFC 1011), which lists all of the available protocols, and also to look at some of the major TCP/IP implementations to see what various vendors have added.

2. General description of the TCP/IP protocols

TCP/IP is a layered set of protocols. In order to understand what this means, it is useful to look at an example. A typical situation is sending mail. First, there is a protocol for mail. This defines a set of commands which one machine sends to another, e.g. commands to specify who the sender of the message is, who it is being sent to, and then the text of the message. However this protocol assumes that there is a way to communicate reliably between the two computers. Mail, like other application protocols, simply defines a set of commands and messages to be sent. It is designed to be used together with TCP and IP. TCP is responsible for making sure that the commands get through to the other end. It keeps track of what is sent, and retransmitts anything that did not get through. If any message is too large for one datagram, e.g. the text of the mail, TCP will split it up into several datagrams, and make sure that they all arrive correctly. Since these functions are needed for many applications, they are put together into a separate protocol, rather than being part of the specifications for sending mail. You can think of TCP as forming a library of routines that applications can use when they need reliable network communications with another computer. Similarly, TCP calls on the services of IP. Although the services that TCP supplies are needed by many applications, there are still some kinds of applications that don't need them. However there are some services that every application needs. So these services are put together into IP. As with TCP, you can think of IP as a library of routines that TCP calls on, but which is also available to applications that don't use TCP. This strategy of building several levels of protocol is called "layering". We think of the applications programs such as mail, TCP, and IP, as being separate "layers", each of which calls on the services of the layer below it. Generally, TCP/IP applications use 4 layers:

- an application protocol such as mail
- a protocol such as TCP that provides services need by many applications
- IP, which provides the basic service of getting datagrams to their destination
- the protocols needed to manage a specific physical medium, such as Ethernet or a point to point line.

TCP/IP is based on the "catenet model". (This is described in more detail in IEN 48.) This model assumes that there are a large number of independent networks connected together by gateways. The user should be able to access computers or other resources on any of these networks. Datagrams will often pass through a dozen different networks before getting to their final destination. The routing needed to accomplish this should be completely invisible to the user. As far as the user is concerned, all he needs to know in order to access another system is an "Internet address". This is an address that looks like 128.6.4.194. It is actually a 32-bit number. However it is normally written as 4 decimal numbers, each representing 8 bits of the address. (The term "octet" is used by Internet documentation for such 8-bit chunks. The term "byte" is not used, because TCP/IP is supported by some computers that have byte sizes other than 8 bits.) Generally the structure of the address gives you some information about how to get to the system. For example, 128.6 is a network number assigned by a central authority to Rutgers University. Rutgers uses the next octet to indicate which of the campus Ethernets is involved. 128.6.4 happens to be an Ethernet used by the Computer Science Department. The last octet allows for up to 254 systems on each Ethernet. (It is 254 because 0 and 255 are not allowed, for reasons that will be discussed later.) Note that 128.6.4.194 and 128.6.5.194 would be different systems. The structure of an Internet address is described in a bit more detail later.

Of course we normally refer to systems by name, rather than by Internet address. When we specify a name, the network software looks it up in a database, and comes up with the corresponding Internet address. Most of the network software deals strictly in terms of the address. (RFC 882 describes the name server technology used to handle this lookup.)

TCP/IP is built on "connectionless" technology. Information is transfered as a sequence of "datagrams". A datagram is a collection of data that is sent as a single message. Each of these datagrams is sent through the network individually. There are provisions to open connections (i.e. to start a conversation that will continue for some time). However at some level, information from those connections is broken up into datagrams, and those datagrams are treated by the network as completely separate. For example, suppose you want to transfer a 15000 octet file. Most networks can't handle a 15000 octet datagram. So the protocols will break this up into something like 30 500-octet datagrams. Each of these datagrams will be sent to the other end. At that point, they will be put back together into the 15000-octet file. However while those datagrams are in transit, the network doesn't know that there is any connection between them. It is perfectly possible that datagram 14 will actually arrive before datagram 13. It is also possible that somewhere in the network, an error will occur, and some datagram won't get through at all. In that case, that datagram has to be sent again.

Note by the way that the terms "datagram" and "packet" often seem to be nearly interchangable. Technically, datagram is the right word to use when describing TCP/IP. A datagram is a unit of data, which is what the protocols deal with. A packet is a physical thing, appearing on an Ethernet or some wire. In most cases a packet simply contains a datagram, so there is very little difference. However they can differ. When TCP/IP is used on top of X.25, the X.25 interface breaks the datagrams up into 128-byte packets. This is invisible to IP, because the packets are put back together into a single datagram at the other end before being processed by TCP/IP. So in this case, one IP datagram would be carried by several packets. However with most media, there are efficiency advantages to sending one datagram per packet, and so the distinction tends to vanish.

2.1. The TCP level

Two separate protocols are involved in handling TCP/IP datagrams. TCP (the "transmission control protocol") is responsible for breaking up the message into datagrams, reassembling them at the other end, resending anything that gets lost, and putting things back in the right order. IP (the "internet protocol") is responsible for routing individual datagrams. It may seem like TCP is doing all the work. And in small networks that is true. However in the Internet, simply getting a datagram to its destination can be a complex job. A connection may require the datagram to go through several networks at Rutgers, a serial line to the John von Neuman Supercomputer Center, a couple of Ethernets there, a series of 56Kbaud phone lines to another NSFnet site, and more Ethernets on another campus. Keeping track of the routes to all of the destinations and handling incompatibilities among different transport media turns out to be a complex job. Note that the interface between TCP and IP is fairly simple. TCP simply hands IP a datagram with a destination. IP doesn't know how this datagram relates to any datagram before it or after it.

It may have occurred to you that something is missing here. We have talked about Internet addresses, but not about how you keep track of multiple connections to a given system. Clearly it isn't enough to get a datagram to the right destination. TCP has to know which connection this datagram is part of. This task is referred to as "demultiplexing." In fact, there are several levels of demultiplexing going on in TCP/IP. The information needed to do this demultiplexing is contained in a series of "headers". A header is simply a few extra octets tacked onto the beginning of a datagram by some protocol in order to keep track of it. It's a lot like putting a letter into an envelope and putting an address on the outside of the envelope. Except with modern networks it happens several times. It's like you put the letter into a little envelope, your secretary puts that into a somewhat bigger envelope, the campus mail center puts that envelope into a still bigger one, etc. Here is an overview of the headers that get stuck on a message that passes through a typical TCP/IP network:

We start with a single data stream, say a file you are trying to send to some other computer:

```
        ......................................................
```

TCP breaks it up into manageable chunks. (In order to do this, TCP has to know how large a datagram your network can handle. Actually, the TCP's at each end say how big a datagram they can handle, and then they pick the smallest size.)

```
        ....   ....   ....   ....   ....   ....   ....   ....
```

TCP puts a header at the front of each datagram. This header actually contains at least 20 octets, but the most important ones are a source and destination "port number" and a "sequence number". The port numbers are used to keep track of different conversations. Suppose 3 different people are transferring files. Your TCP might allocate port numbers 1000, 1001, and 1002 to these transfers. When you are sending a datagram, this becomes the "source" port number, since you are the source of the datagram. Of course the TCP at the other end has assigned a port number of its own for the conversation. Your TCP has to know the port number used by the other end as well. (It finds out when the connection starts, as we will explain below.) It puts this in the "destination" port field. Of course if the other end sends a datagram back to you, the source and destination port numbers will be reversed, since then it will be the source and you will be the destination. Each datagram has a sequence number. This is used so that the other end can make sure that it gets the datagrams in the right order, and that it hasn't missed any. (See the TCP specification for details.) TCP doesn't number the datagrams, but the octets. So if there are 500 octets of data in each datagram, the first datagram might be numbered 0, the second 500, the next 1000, the next 1500, etc. Finally, I will mention the Checksum. This is a number that is computed by adding up all the octets in the datagram (more or less - see the TCP spec). The result is put in the header. TCP at the other end computes the checksum again. If they disagree, then something bad happened to the datagram in transmission, and it is thrown away. So here's what the datagram looks like now.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   your data ... next 500 octets                               |
|   ......                                                      |
```

If we abbreviate the TCP header as "T", the whole file now looks like this:

```
T....   T....   T....   T....   T....   T....   T....
```

You will note that there are items in the header that I have not described above. They are generally involved with managing the connection. In order to make sure the datagram has arrived at its destination, the recipient has to send back an "acknowledgement". This is a datagram whose "Acknowledgement number" field is filled in. For example, sending a packet with an acknowledgement of 1500 indicates that you have received all the data up to octet number 1500. If the sender doesn't get an acknowledgement within a reasonable amount of time, it sends the data again. The window is used to control how much data can be in transit at any one time. It is not practical to wait for each datagram to be acknowledged before sending the next one. That would slow things down too much. On the other hand, you can't just keep sending, or a fast computer might overrun the capacity of a slow one to absorb data. Thus each end indicates how much new data it is currently prepared to absorb by putting the number of octets in its "Window" field. As the computer receives data, the amount of space left in its window decreases. When it goes to zero, the sender has to stop. As the receiver processes the data, it increases its window, indicating that it is ready to accept more data. Often the same datagram can be used to acknowledge receipt of a set of data and to give permission for additional new data (by an updated window). The "Urgent" field allows one end to tell the other to skip ahead in its processing to a particular octet. This is often useful for handling asynchronous events, for example when you type a control character or other command that interrupts output. The other fields are beyond the scope of this document.

2.2. The IP level

TCP sends each of these datagrams to IP. Of course it has to tell IP the Internet address of the computer at the other end. Note that this is all IP is concerned about. It doesn't care about what is in the datagram, or even in the TCP header. IP's job is simply to find a route for the datagram and get it to the other end. In order to allow gateways or other intermediate systems to forward the datagram, it adds its own header. The main things in this header are the source and destination Internet address (32-bit addresses, like 128.6.4.194), the protocol number, and another checksum. The source Internet address is simply the address of your machine. (This is necessary so the other end knows where the datagram came from.) The destination Internet address is the address of the other machine. (This is necessary so any gateways in the middle know where you want the datagram to go.) The protocol number tells IP at the other end to send the datagram to TCP. Although most IP traffic uses TCP, there are other protocols that can use IP, so you have to tell IP which protocol to send the datagram to. Finally, the checksum allows IP at the other end to verify that the header wasn't damaged in transit. Note that TCP and IP have separate checksums. IP needs to be able to verify that the header didn't get damaged in transit, or it could send a message to the wrong place. For reasons not worth discussing here, it is both more efficient and safer to have TCP compute a separate checksum for the TCP header and data. Once IP has tacked on its header, here's what the message looks like:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  TCP header, then your data ......                            |
|                                                               |
```

If we represent the IP header by an "I", your file now looks like this:

```
IT....   IT....   IT....   IT....   IT....   IT....   IT....
```

Again, the header contains some additional fields that have not been discussed. Most of them are beyond the scope of this document. The flags and fragment offset are used to keep track of the pieces when a datagram has to be split up. This can happen when datagrams are forwarded through a network for which they are too big. (This will be discussed a bit more below.) The time to live is a number that is decremented whenever the datagram passes through a system. When it goes to zero, the datagram is discarded. This is done in case a loop develops in the system somehow. Of course this should be impossible, but well-designed networks are built to cope with "impossible" conditions.

At this point, it's possible that no more headers are needed. If your computer happens to have a direct phone line connecting it to the destination computer, or to a gateway, it may simply send the datagrams out on the line (though likely a synchronous protocol such as HDLC would be used, and it would add at least a few octets at the beginning and end).

2.3. The Ethernet level

However most of our networks these days use Ethernet. So now we have to describe Ethernet's headers. Unfortunately, Ethernet has its own addresses. The people who designed Ethernet wanted to make sure that no two machines would end up with the same Ethernet address. Furthermore, they didn't want the user to have to worry about assigning addresses. So each Ethernet controller comes with an address builtin from the factory. In order to make sure that they would never have to reuse addresses, the Ethernet designers allocated 48 bits for the Ethernet address. People who make Ethernet equipment have to register with a central authority, to make sure that the numbers they assign don't overlap any other manufacturer. Ethernet is a "broadcast medium". That is, it is in effect like an old party line telephone. When you send a packet out on the Ethernet, every machine on the network sees the packet. So something is needed to make sure that the right machine gets it. As you might guess, this involves the Ethernet header. Every Ethernet packet has a 14-octet header that includes the source and destination Ethernet address, and a type code. Each machine is supposed to pay attention only to packets with its own Ethernet address in the destination field. (It's perfectly possible to cheat, which is one reason that Ethernet communications are not terribly secure.) Note that there is no connection between the Ethernet address and the Internet address. Each machine has to have a table of what Ethernet address corresponds to what Internet address. (We will describe how this table is constructed a bit later.) In addition to the addresses, the header contains a type code. The type code is to allow for several different protocol families to be used on the same network. So you can use TCP/IP, DECnet, Xerox NS, etc. at the same time. Each of them will put a different value in the type field. Finally, there is a checksum. The Ethernet controller computes a checksum of the entire packet. When the other end receives the packet, it recomputes the checksum, and throws the packet away if the answer disagrees with the original. The checksum is put on the end of the packet, not in the header.

The final result is that your message looks like this:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Ethernet destination address (first 32 bits)            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Ethernet dest (last 16 bits)  |Ethernet source (first 16 bits)|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Ethernet source address (last 32 bits)                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Type code              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  IP header, then TCP header, then your data                   |
|                                                               |
    ...
|                                                               |
|   end of your data                                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Ethernet Checksum                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

If we represent the Ethernet header with "E", and the Ethernet checksum with "C", your file now looks like this:

```
EIT....C   EIT....C   EIT....C   EIT....C   EIT....C
```

When these packets are received by the other end, of course all the headers are removed. The Ethernet interface removes the Ethernet header and the checksum. It looks at the type code. Since the type code is the one assigned to IP, the Ethernet device driver passes the datagram up to IP. IP removes the IP header. It looks at the IP protocol field. Since the protocol type is TCP, it passes the datagram up to TCP. TCP now looks at the sequence number. It uses the sequence numbers and other information to combine all the datagrams into the original file.

The ends our initial summary of TCP/IP. There are still some crucial concepts we haven't gotten to, so we'll now go back and add details in several areas. (For detailed descriptions of the items discussed here see, RFC 793 for TCP, RFC 791 for IP, and RFC's 894 and 826 for sending IP over Ethernet.)

3. Well-known sockets and the applications layer

So far, we have described how a stream of data is broken up into datagrams, sent to another computer, and put back together. However something more is needed in order to accomplish anything useful. There has to be a way for you to open a connection to a specified computer, log into it, tell it what file you want, and control the transmission of the file. (If you have a different application in mind, e.g. computer mail, some analogous protocol is needed.) This is done by "application protocols". The application protocols run "on top" of TCP/IP. That is, when they want to send a message, they give the message to TCP. TCP makes sure it gets delivered to the other end. Because TCP and IP take care of all the networking details, the applications protocols can treat a network connection as if it were a simple byte stream, like a terminal or phone line.

Before going into more details about applications programs, we have to describe how you find an application. Suppose you want to send a file to a computer whose Internet address is 128.6.4.7. To start the process, you need more than just the Internet address. You have to connect to the FTP server at the other end. In general, network programs are specialized for a specific set of tasks. Most systems have separate programs to handle file transfers, remote terminal logins, mail, etc. When you connect to 128.6.4.7, you have to specify that you want to talk to the FTP server. This is done by having "well-known sockets" for each server. Recall that TCP uses port numbers to keep track of individual conversations. User programs normally use more or less random port numbers. However specific port numbers are assigned to the programs that sit waiting for requests. For example, if you want to send a file, you will start a program called "ftp". It will open a connection using some random number, say 1234, for the port number on its end. However it will specify port number 21 for the other end. This is the official port number for the FTP server. Note that there are two different programs involved. You run ftp on your side. This is a program designed to accept commands from your terminal and pass them on to the other end. The program that you talk to on the other machine is the FTP server. It is designed to accept commands from the network connection, rather than an interactive terminal. There is no need for your program to use a well-known socket number for itself. Nobody is trying to find it. However the servers have to have well-known numbers, so that people can open connections to them and start sending them commands. The official port numbers for each program are given in "Assigned Numbers".

Note that a connection is actually described by a set of 4 numbers: the Internet address at each end, and the TCP port number at each end. Every datagram has all four of those numbers in it. (The Internet addresses are in the IP header, and the TCP port numbers are in the TCP header.) In order to keep things straight, no two connections can have the same set of numbers. However it is enough for any one number to be different. For example, it is perfectly possible for two different users on a machine to be sending files to the same other machine. This could result in connections with the following parameters:

	Internet addresses	TCP ports
connection 1	128.6.4.194, 128.6.4.7	1234, 21
connection 2	128.6.4.194, 128.6.4.7	1235, 21

Since the same machines are involved, the Internet addresses are the same. Since they are both doing file transfers, one end of the connection involves the well-known port number for FTP. The only thing that differs is the port number for the program that the users are running. That's enough of a difference. Generally, at least one end of the connection asks the network software to assign it a port number that is guaranteed to be unique. Normally, it's the user's end, since the server has to use a well-known number.

Now that we know how to open connections, let's get back to the applications programs. As mentioned earlier, once TCP has opened a connection, we have something that might as well be a simple wire. All the hard parts are handled by TCP and IP. However we still need some agreement as to what we send over this connection. In effect this is simply an agreement on what set of commands the application will understand, and the format in which they are to be sent. Generally, what is sent is a combination of commands and data. They use context to differentiate. For example, the mail protocol works like this: Your mail program opens a connection to the mail server at the other end. Your program gives it your machine's name, the sender of the message, and the recipients you want it sent to. It then sends a command saying that it is starting the message. At that point, the other end stops treating what it sees as commands, and starts accepting the message. Your end then starts sending the text of the message. At the end of the message, a special mark is sent (a dot in the first column). After that, both ends understand that your program is again sending commands. This is the simplest way to do things, and the one that most applications use.

File transfer is somewhat more complex. The file transfer protocol involves two different connections. It starts out just like mail. The user's program sends commands like "log me in as this user", "here is my password", "send me the file with this name". However once the command to send data is sent, a second connection is opened for the data itself. It would certainly be possible to send the data on the same connection, as mail does. However file transfers often take a long time. The designers of the file transfer protocol wanted to allow the user to continue issuing commands while the transfer is going on. For example, the user might make an inquiry, or he might abort the transfer. Thus the designers felt it was best to use a separate connection for the data and leave the original command connection for commands. (It is also possible to open command connections to two different computers, and tell them to send a file from one to the other. In that case, the data couldn't go over the command connection.)

Remote terminal connections use another mechanism still. For remote logins, there is just one connection. It normally sends data. When it is necessary to send a command (e.g. to set the terminal type or to change some mode), a special character is used to indicate that the next character is a command. If the user happens to type that special character as data, two of them are sent.

We are not going to describe the application protocols in detail in this document. It's better to read the RFC's yourself. However there are a couple of common conventions used by applications that will be described here. First, the common network representation: TCP/IP is intended to be usable on any computer. Unfortunately, not all computers agree on how data is represented. There are differences in character codes (ASCII vs. EBCDIC), in end of line conventions (carriage return, line feed, or a representation using counts), and in whether terminals expect characters to be sent individually or a line at a time. In order to allow computers of different kinds to communicate, each applications protocol defines a standard representation. Note that TCP and IP do not care about the representation. TCP simply sends octets. However the programs at both ends have to agree on how the octets are to be interpreted. The RFC for each application specifies the standard representation for that application. Normally it is "net ASCII". This uses ASCII characters, with end of line denoted by a carriage return followed by a line feed. For remote login, there is also a definition of a "standard terminal", which turns out to be a half-duplex terminal with echoing happening on the local machine. Most applications also make provisions for the two computers to agree on other representations that they may find more convenient. For example, PDP-10's have 36-bit words. There is a way that two PDP-10's can agree to send a 36-bit binary file. Similarly, two systems that prefer full-duplex terminal conversations can agree on that. However each application has a standard representation, which every machine must support.

3.1. An example application: SMTP

In order to give a bit better idea what is involved in the application protocols, I'm going to show an example of SMTP, which is the mail protocol. (SMTP is "simple mail transfer protocol.) We assume that a computer called TOPAZ.RUTGERS.EDU wants to send the following message.

```
Date: Sat, 27 Jun 87 13:26:31 EDT
From: hedrick@topaz.rutgers.edu
To: levy@red.rutgers.edu
Subject: meeting

Let's get together Monday at 1pm.
```

First, note that the format of the message itself is described by an Internet standard (RFC 822). The standard specifies the fact that the message must be transmitted as net ASCII (i.e. it must be ASCII, with carriage return/linefeed to delimit lines). It also describes the general structure, as a group of header lines, then a blank line, and then the body of the message. Finally, it describes the syntax of the header lines in detail. Generally they consist of a keyword and then a value.

Note that the addressee is indicated as LEVY@RED.RUTGERS.EDU. Initially, addresses were simply "person at machine". However recent standards have made things more flexible. There are now provisions for systems to handle other systems' mail. This can allow automatic forwarding on behalf of computers not connected to the Internet. It can be used to direct mail for a number of systems to one central mail server. Indeed there is no requirement that an actual computer by the name of RED.RUTGERS.EDU even exist. The name servers could be set up so that you mail to department names, and each department's mail is routed automatically to an appropriate computer. It is also possible that the part before the @ is something other than a user name. It is possible for programs to be set up to process mail. There are also provisions to handle mailing lists, and generic names such as "postmaster" or "operator".

The way the message is to be sent to another system is described by RFC's 821 and 974. The program that is going to be doing the sending asks the name server several queries to determine where to route the message. The first query is to find out which machines handle mail for the name RED.RUTGERS.EDU. In this case, the server replies that RED.RUTGERS.EDU handles its own mail. The program then asks for the address of RED.RUTGERS.EDU, which is 128.6.4.2. Then the mail program opens a TCP connection to port 25 on 128.6.4.2. Port 25 is the well-known socket used for receiving mail. Once this connection is established, the mail program starts sending commands. Here is a typical conversation. Each line is labelled as to whether it is from TOPAZ or RED. Note that TOPAZ initiated the connection:

```
RED    220 RED.RUTGERS.EDU SMTP Service at 29 Jun 87 05:17:18 EDT
TOPAZ  HELO topaz.rutgers.edu
RED    250 RED.RUTGERS.EDU - Hello, TOPAZ.RUTGERS.EDU
TOPAZ  MAIL From:<hedrick@topaz.rutgers.edu>
RED    250 MAIL accepted
TOPAZ  RCPT To:<levy@red.rutgers.edu>
RED    250 Recipient accepted
TOPAZ  DATA
RED    354 Start mail input; end with <CRLF>.<CRLF>
TOPAZ  Date: Sat, 27 Jun 87 13:26:31 EDT
TOPAZ  From: hedrick@topaz.rutgers.edu
TOPAZ  To: levy@red.rutgers.edu
TOPAZ  Subject: meeting
TOPAZ
TOPAZ  Let's get together Monday at 1pm.
TOPAZ  .
RED    250 OK
TOPAZ  QUIT
RED    221 RED.RUTGERS.EDU Service closing transmission channel
```

First, note that commands all use normal text. This is typical of the Internet standards. Many of the protocols use standard ASCII commands. This makes it easy to watch what is going on and to diagnose problems. For example, the mail program keeps a log of each conversation. If something goes wrong, the log file can simply be mailed to the postmaster. Since it is normal text, he can see what was going on. It also allows a human to interact directly with the mail server, for testing. (Some newer protocols are complex enough that this is not practical. The commands would have to have a syntax that would require a significant parser. Thus there is a tendency for newer protocols to use binary formats. Generally they are structured like C or Pascal record structures.) Second, note that the responses all begin with numbers. This is also typical of Internet protocols. The allowable responses are defined in the protocol. The numbers allow the user program to respond unambiguously. The rest of the response is text, which is normally for use by any human who may be watching or looking at a log. It has no effect on the operation of the programs. (However there is one point at which the protocol uses part of the text of the response.) The commands themselves simply allow the mail program on one end to tell the mail server the information it needs to know in order to deliver the message. In this case, the mail server could get the information by looking at the message itself. But for more complex cases, that would not be safe. Every session must begin with a HELO, which gives the name of the system that initiated the connection. Then the sender and recipients are specified. (There can be more than one RCPT command, if there are several recipients.) Finally the data itself is sent. Note that the text of the message is terminated by a line containing just a period. (If such a line appears in the message, the period is doubled.) After the message is accepted, the sender can send another message, or terminate the session as in the example above.

Generally, there is a pattern to the response numbers. The protocol defines the specific set of responses that can be sent as answers to any given command. However programs that don't want to analyze them in detail can just look at the first digit. In general, responses that begin with a 2 indicate success. Those that begin with 3 indicate that some further action is needed, as shown above. 4 and 5 indicate errors. 4 is a "temporary" error, such as a disk filling. The message should be saved, and tried again later. 5 is a permanent error, such as a nonexistent recipient. The message should be returned to the sender with an error message.

(For more details about the protocols mentioned in this section, see RFC's 821/822 for mail, RFC 959 for file transfer, and RFC's 854/855 for remote logins. For the well-known port numbers, see the current edition of Assigned Numbers, and possibly RFC 814.)

4. Protocols other than TCP: UDP and ICMP

So far, we have described only connections that use TCP. Recall that TCP is responsible for breaking up messages into datagrams, and reassembling them properly. However in many applications, we have messages that will always fit in a single datagram. An example is name lookup. When a user attempts to make a connection to another system, he will generally specify the system by name, rather than Internet address. His system has to translate that name to an address before it can do anything. Generally, only a few systems have the database used to translate names to addresses. So the user's system will want to send a query to one of the systems that has the database. This query is going to be very short. It will certainly fit in one datagram. So will the answer. Thus it seems silly to use TCP. Of course TCP does more than just break things up into datagrams. It also makes sure that the data arrives, resending datagrams where necessary. But for a question that fits in a single datagram, we don't need all the complexity of TCP to do this. If we don't get an answer after a few seconds, we can just ask again. For applications like this, there are alternatives to TCP.

The most common alternative is UDP ("user datagram protocol"). UDP is designed for applications where you don't need to put sequences of datagrams together. It fits into the system much like TCP. There is a UDP header. The network software puts the UDP header on the front of your data, just as it would put a TCP header on the front of your data. Then UDP sends the data to IP, which adds the IP header, putting UDP's protocol number in the protocol field instead of TCP's protocol number. However UDP doesn't do as much as TCP does. It doesn't split data into multiple datagrams. It doesn't keep track of what it has sent so it can resend if necessary. About all that UDP provides is port numbers, so that several programs can use UDP at once. UDP port numbers are used just like TCP port numbers. There are well-known port numbers for servers that use UDP. Note that the UDP header is shorter than a TCP header. It still has source and destination port numbers, and a checksum, but that's about it. No sequence number, since it is not needed. UDP is used by the protocols that handle name lookups (see IEN 116, RFC 882, and RFC 883), and a number of similar protocols.

Another alternative protocol is ICMP ("Internet control message protocol"). ICMP is used for error messages, and other messages intended for the TCP/IP software itself, rather than any particular user program. For example, if you attempt to connect to a host, your system may get back an ICMP message saying "host unreachable". ICMP can also be used to find out some information about the network. See RFC 792 for details of ICMP. ICMP is similar to UDP, in that it handles messages that fit in one datagram. However it is even simpler than UDP. It doesn't even have port numbers in its header. Since all ICMP messages are interpreted by the network software itself, no port numbers are needed to say where a ICMP message is supposed to go.

5. Keeping track of names and information: the domain system

As we indicated earlier, the network software generally needs a 32-bit Internet address in order to open a connection or send a datagram. However users prefer to deal with computer names rather than numbers. Thus there is a database that allows the software to look up a name and find the corresponding number. When the Internet was small, this was easy. Each system would have a file that listed all of the other systems, giving both their name and number. There are now too many computers for this approach to be practical. Thus these files have been replaced by a set of name servers that keep track of host names and the corresponding Internet addresses. (In fact these servers are somewhat more general than that. This is just one kind of information stored in the domain system.) Note that a set of interlocking servers are used, rather than a single central one. There are now so many different institutions connected to the Internet that it would be impractical for them to notify a central authority whenever they installed or moved a computer. Thus naming authority is delegated to individual institutions. The name servers form a tree, corresponding to institutional structure. The names themselves follow a similar structure. A typical example is the name BORAX.LCS.MIT.EDU. This is a computer at the Laboratory for Computer Science (LCS) at MIT. In order to find its Internet address, you might potentially have to consult 4 different servers. First, you would ask a central server (called the root) where the EDU server is. EDU is a server that keeps track of educational institutions. The root server would give you the names and Internet addresses of several servers for EDU. (There are several servers at each level, to allow for the possibly that one might be down.) You would then ask EDU where the server for MIT is. Again, it would give you names and Internet addresses of several servers for MIT. Generally, not all of those servers would be at MIT, to allow for the possibility of a general power failure at MIT. Then you would ask MIT where the server for LCS is, and finally you would ask one of the LCS servers about BORAX. The final result would be the Internet address for BORAX.LCS.MIT.EDU. Each of these levels is referred to as a "domain". The entire name, BORAX.LCS.MIT.EDU, is called a "domain name". (So are the names of the higher-level domains, such as LCS.MIT.EDU, MIT.EDU, and EDU.)

Fortunately, you don't really have to go through all of this most of the time. First of all, the root name servers also happen to be the name servers for the top-level domains such as EDU. Thus a single query to a root server will get you to MIT. Second, software generally remembers answers that it got before. So once we look up a name at LCS.MIT.EDU, our software remembers where to find servers for LCS.MIT.EDU, MIT.EDU, and EDU. It also remembers the translation of BORAX.LCS.MIT.EDU. Each of these pieces of information has a "time to live" associated with it. Typically this is a few days. After that, the information expires and has to be looked up again. This allows institutions to change things.

The domain system is not limited to finding out Internet addresses. Each domain name is a node in a database. The node can have records that define a number of different properties. Examples are Internet address, computer type, and a list of services provided by a computer. A program can ask for a specific piece of information, or all information about a given name. It is possible for a node in the database to be marked as an "alias" (or nickname) for another node. It is also possible to use the domain system to store information about users, mailing lists, or other objects.

There is an Internet standard defining the operation of these databases, as well as the protocols used to make queries of them. Every network utility has to be able to make such queries, since this is now the official way to evaluate host names. Generally utilities will talk to a server on their own system. This server will take care of contacting the other servers for them. This keeps down the amount of code that has to be in each application program.

The domain system is particularly important for handling computer mail. There are entry types to define what computer handles mail for a given name, to specify where an individual is to receive mail, and to define mailing lists.

(See RFC's 882, 883, and 973 for specifications of the domain system. RFC 974 defines the use of the domain system in sending mail.)

6. Routing

The description above indicated that the IP implementation is responsible for getting datagrams to the destination indicated by the destination address, but little was said about how this would be done. The task of finding how to get a datagram to its destination is referred to as "routing". In fact many of the details depend upon the particular implementation. However some general things can be said.

First, it is necessary to understand the model on which IP is based. IP assumes that a system is attached to some local network. We assume that the system can send datagrams to any other system on its own network. (In the case of Ethernet, it simply finds the Ethernet address of the destination system, and puts the datagram out on the Ethernet.) The problem comes when a system is asked to send a datagram to a system on a different network. This problem is handled by gateways. A gateway is a system that connects a network with one or more other networks. Gateways are often normal computers that happen to have more than one network interface. For example, we have a Unix machine that has two different Ethernet interfaces. Thus it is connected to networks 128.6.4 and 128.6.3. This machine can act as a gateway between those two networks. The software on that machine must be set up so that it will forward datagrams from one network to the other. That is, if a machine on network 128.6.4 sends a datagram to the gateway, and the datagram is addressed to a machine on network 128.6.3, the gateway will forward the datagram to the destination. Major communications centers often have gateways that connect a number of different networks. (In many cases, special-purpose gateway systems provide better performance or reliability than general-purpose systems acting as gateways. A number of vendors sell such systems.)

Routing in IP is based entirely upon the network number of the destination address. Each computer has a table of network numbers. For each network number, a gateway is listed. This is the gateway to be used to get to that network. Note that the gateway doesn't have to connect directly to the network. It just has to be the best place to go to get there. For example at Rutgers, our interface to NSFnet is at the John von Neuman Supercomputer Center (JvNC). Our connection to JvNC is via a high-speed serial line connected to a gateway whose address is 128.6.3.12. Systems on net 128.6.3 will list 128.6.3.12 as the gateway for many off-campus networks. However systems on net 128.6.4 will list 128.6.4.1 as the gateway to those same off-campus networks. 128.6.4.1 is the gateway between networks 128.6.4 and 128.6.3, so it is the first step in getting to JvNC.

When a computer wants to send a datagram, it first checks to see if the destination address is on the system's own local network. If so, the datagram can be sent directly. Otherwise, the system expects to find an entry for the network that the destination address is on. The datagram is sent to the gateway listed in that entry. This table can get quite big. For example, the Internet now includes several hundred individual networks. Thus various strategies have been developed to reduce the size of the routing table. One strategy is to depend upon "default routes". Often, there is only one gateway out of a network. This gateway might connect a local Ethernet to a campus-wide backbone network. In that case, we don't need to have a separate entry for every network in the world. We simply define that gateway as a "default". When no specific route is found for a datagram, the datagram is sent to the default gateway. A default gateway can even be used when there are several gateways on a network. There are provisions for gateways to send a message saying "I'm not the best gateway -- use this one instead." (The message is sent via ICMP. See RFC 792.) Most network software is designed to use these messages to add entries to their routing tables. Suppose network 128.6.4 has two gateways, 128.6.4.59 and 128.6.4.1. 128.6.4.59 leads to several other internal Rutgers networks. 128.6.4.1 leads indirectly to the NSFnet. Suppose we set 128.6.4.59 as a default gateway, and have no other routing table entries. Now what happens when we need to send a datagram to MIT? MIT is network 18. Since we have no entry for network 18, the datagram will be sent to the default, 128.6.4.59. As it happens, this gateway is the wrong one. So it will forward the datagram to 128.6.4.1. But it will also send back an error saying in effect: "to get to network 18, use 128.6.4.1". Our software will then add an entry to the routing table. Any future datagrams to MIT will then go directly to 128.6.4.1. (The error message is sent using the ICMP protocol. The message type is called "ICMP redirect.")

Most IP experts recommend that individual computers should not try to keep track of the entire network. Instead, they should start with default gateways, and let the gateways tell them the routes, as just described. However this doesn't say how the gateways should find out about the routes. The gateways can't depend upon this strategy. They have to have fairly complete routing tables. For this, some sort of routing protocol is needed. A routing protocol is simply a technique for the gateways to find each other, and keep up to date about the best way to get to every network. RFC 1009 contains a review of gateway design and routing. However rip.doc is probably a better introduction to the subject. It contains some tutorial material, and a detailed description of the most commonly-used routing protocol.

7. Details about Internet addresses: subnets and broadcasting

As indicated earlier, Internet addresses are 32-bit numbers, normally written as 4 octets (in decimal), e.g. 128.6.4.7. There are actually 3 different types of address. The problem is that the address has to indicate both the network and the host within the network. It was felt that eventually there would be lots of networks. Many of them would be small, but probably 24 bits would be needed to represent all the IP networks. It was also felt that some very big networks might need 24 bits to represent all of their hosts. This would seem to lead to 48 bit addresses. But the designers really wanted to use 32 bit addresses. So they adopted a kludge. The assumption is that most of the networks will be small. So they set up three different ranges of address. Addresses beginning with 1 to 126 use only the first octet for the network number. The other three octets are available for the host number. Thus 24 bits are available for hosts. These numbers are used for large networks. But there can only be 126 of these very big networks. The Arpanet is one, and there are a few large commercial networks. But few normal organizations get one of these "class A" addresses. For normal large organizations, "class B" addresses are used. Class B addresses use the first two octets for the network number. Thus network numbers are 128.1 through 191.254. (We avoid 0 and 255, for reasons that we see below. We also avoid addresses beginning with 127, because that is used by some systems for special purposes.) The last two octets are available for host addesses, giving 16 bits of host address. This allows for 64516 computers, which should be enough for most organizations. (It is possible to get more than one class B address, if you run out.) Finally, class C addresses use three octets, in the range 192.1.1 to 223.254.254. These allow only 254 hosts on each network, but there can be lots of these networks. Addresses above 223 are reserved for future use, as class D and E (which are currently not defined).

Many large organizations find it convenient to divide their network number into "subnets". For example, Rutgers has been assigned a class B address, 128.6. We find it convenient to use the third octet of the address to indicate which Ethernet a host is on. This division has no significance outside of Rutgers. A computer at another institution would treat all datagrams addressed to 128.6 the same way. They would not look at the third octet of the address. Thus computers outside Rutgers would not have different routes for 128.6.4 or 128.6.5. But inside Rutgers, we treat 128.6.4 and 128.6.5 as separate networks. In effect, gateways inside Rutgers have separate entries for each Rutgers subnet, whereas gateways outside Rutgers just have one entry for 128.6. Note that we could do exactly the same thing by using a separate class C address for each Ethernet. As far as Rutgers is concerned, it would be just as convenient for us to have a number of class C addresses. However using class C addresses would make things inconvenient for the rest of the world. Every institution that wanted to talk to us would have to have a separate entry for each one of our networks. If every institution did this, there would be far too many networks for any reasonable gateway to keep track of. By subdividing a class B network, we hide our internal structure from everyone else, and save them trouble. This subnet strategy requires special provisions in the network software. It is described in RFC 950.

0 and 255 have special meanings. 0 is reserved for machines that don't know their address. In certain circumstances it is possible for a machine not to know the number of the network it is on, or even its own host address. For example, 0.0.0.23 would be a machine that knew it was host number 23, but didn't know on what network.

255 is used for "broadcast". A broadcast is a message that you want every system on the network to see. Broadcasts are used in some situations where you don't know who to talk to. For example, suppose you need to look up a host name and get its Internet address. Sometimes you don't know the address of the nearest name server. In that case, you might send

the request as a broadcast. There are also cases where a number of systems are interested in information. It is then less expensive to send a single broadcast than to send datagrams individually to each host that is interested in the information. In order to send a broadcast, you use an address that is made by using your network address, with all ones in the part of the address where the host number goes. For example, if you are on network 128.6.4, you would use 128.6.4.255 for broadcasts. How this is actually implemented depends upon the medium. It is not possible to send broadcasts on the Arpanet, or on point to point lines. However it is possible on an Ethernet. If you use an Ethernet address with all its bits on (all ones), every machine on the Ethernet is supposed to look at that datagram.

Although the official broadcast address for network 128.6.4 is now 128.6.4.255, there are some other addresses that may be treated as broadcasts by certain implementations. For convenience, the standard also allows 255.255.255.255 to be used. This refers to all hosts on the local network. It is often simpler to use 255.255.255.255 instead of finding out the network number for the local network and forming a broadcast address such as 128.6.4.255. In addition, certain older implementations may use 0 instead of 255 to form the broadcast address. Such implementations would use 128.6.4.0 instead of 128.6.4.255 as the broadcast address on network 128.6.4. Finally, certain older implementations may not understand about subnets. Thus they consider the network number to be 128.6. In that case, they will assume a broadcast address of 128.6.255.255 or 128.6.0.0. Until support for broadcasts is implemented properly, it can be a somewhat dangerous feature to use.

Because 0 and 255 are used for unknown and broadcast addresses, normal hosts should never be given addresses containing 0 or 255. Addresses should never begin with 0, 127, or any number above 223. Addresses violating these rules are sometimes referred to as "Martians", because of rumors that the Central University of Mars is using network 225.

8. Datagram fragmentation and reassembly

TCP/IP is designed for use with many different kinds of network. Unfortunately, network designers do not agree about how big packets can be. Ethernet packets can be 1500 octets long. Arpanet packets have a maximum of around 1000 octets. Some very fast networks have much larger packet sizes. At first, you might think that IP should simply settle on the smallest possible size. Unfortunately, this would cause serious performance problems. When transferring large files, big packets are far more efficient than small ones. So we want to be able to use the largest packet size possible. But we also want to be able to handle networks with small limits. There are two provisions for this. First, TCP has the ability to "negotiate" about datagram size. When a TCP connection first opens, both ends can send the maximum datagram size they can handle. The smaller of these numbers is used for the rest of the connection. This allows two implementations that can handle big datagrams to use them, but also lets them talk to implementations that can't handle them. However this doesn't completely solve the problem. The most serious problem is that the two ends don't necessarily know about all of the steps in between. For example, when sending data between Rutgers and Berkeley, it is likely that both computers will be on Ethernets. Thus they will both be prepared to handle 1500-octet datagrams. However the connection will at some point end up going over the Arpanet. It can't handle packets of that size. For this reason, there are provisions to split datagrams up into pieces. (This is referred to as "fragmentation".) The IP header contains fields indicating the a datagram has been split, and enough information to let the pieces be put back together. If a gateway connects an Ethernet to the Arpanet, it must be prepared to take 1500-octet Ethernet packets and split them into pieces that will fit on the Arpanet. Furthermore, every host implementation of TCP/IP must be prepared to accept pieces and put them back together. This is referred to as "reassembly".

TCP/IP implementations differ in the approach they take to deciding on datagram size. It is fairly common for implementations to use 576-byte datagrams whenever they can't verify that the entire path is able to handle larger packets. This rather conservative strategy is used because of the number of implementations with bugs in the code to reassemble fragments. Implementors often try to avoid ever having fragmentation occur. Different implementors take different approaches to deciding when it is safe to use large datagrams. Some use them only for the local network. Others will use them for any network on the same campus. 576 bytes is a "safe" size, which every implementation must support.

9. Ethernet encapsulation: ARP

There was a brief discussion earlier about what IP datagrams look like on an Ethernet. The discussion showed the Ethernet header and checksum. However it left one hole: It didn't say how to figure out what Ethernet address to use when you want to talk to a given Internet address. In fact, there is a separate protocol for this, called ARP ("address resolution protocol"). (Note by the way that ARP is not an IP protocol. That is, the ARP datagrams do not have IP headers.) Suppose you are on system 128.6.4.194 and you want to connect to system 128.6.4.7. Your system will first verify that 128.6.4.7 is on the same network, so it can talk directly via Ethernet. Then it will look up 128.6.4.7 in its ARP table, to see if it already knows the Ethernet address. If so, it will stick on an Ethernet header, and send the packet. But suppose this system is not in the ARP table. There is no way to send the packet, because you need the Ethernet address. So it uses the ARP protocol to send an ARP request. Essentially an ARP request says "I need the Ethernet address for 128.6.4.7". Every system listens to ARP requests. When a system sees an ARP request for itself, it is required to respond. So 128.6.4.7 will see the request, and will respond with an ARP reply saying in effect "128.6.4.7 is 8:0:20:1:56:34". (Recall that Ethernet addresses are 48 bits. This is 6 octets. Ethernet addresses are conventionally shown in hex, using the punctuation shown.) Your system will save this information in its ARP table, so future packets will go directly. Most systems treat the ARP table as a cache, and clear entries in it if they have not been used in a certain period of time.

Note by the way that ARP requests must be sent as "broadcasts". There is no way that an ARP request can be sent directly to the right system. After all, the whole reason for sending an ARP request is that you don't know the Ethernet address. So an Ethernet address of all ones is used, i.e. ff:ff:ff:ff:ff:ff. By convention, every machine on the Ethernet is required to pay attention to packets with this as an address. So every system sees every ARP requests. They all look to see whether the request is for their own address. If so, they respond. If not, they could just ignore it. (Some hosts will use ARP requests to update their knowledge about other hosts on the network, even if the request isn't for them.) Note that packets whose IP address indicates broadcast (e.g. 255.255.255.255 or 128.6.4.255) are also sent with an Ethernet address that is all ones.

10. Getting more information

This directory contains documents describing the major protocols. There are literally hundreds of documents, so we have chosen the ones that seem most important. Internet standards are called RFC's. RFC stands for Request for Comment. A proposed standard is initially issued as a proposal, and given an RFC number. When it is finally accepted, it is added to Official Internet Protocols, but it is still referred to by the RFC number. We have also included two IEN's. (IEN's used to be a separate classification for more informal documents. This classification no longer exists -- RFC's are now used for all official Internet documents, and a mailing list is used for more informal reports.) The convention is that whenever an RFC is revised, the revised version gets a new number. This is fine for most purposes, but it causes problems with two documents: Assigned Numbers and Official Internet Protocols. These documents are being revised all the time, so the RFC number keeps changing. You will have to look in rfc-index.txt to find the number of the latest edition. Anyone who is seriously interested in TCP/IP should read the RFC describing IP (791). RFC 1009 is also useful. It is a specification for gateways to be used by NSFnet. As such, it contains an overview of a lot of the TCP/IP technology. You should probably also read the description of at least one of the application protocols, just to get a feel for the way things work. Mail is probably a good one (821/822). TCP (793) is of course a very basic specification. However the spec is fairly complex, so you should only read this when you have the time and patience to think about it carefully. Fortunately, the author of the major RFC's (Jon Postel) is a very good writer. The TCP RFC is far easier to read than you would expect, given the complexity of what it is describing. You can look at the other RFC's as you become curious about their subject matter.

Here is a list of the documents you are more likely to want:

rfc-index	list of all RFC's
rfc1012	somewhat fuller list of all RFC's
rfc1011	Official Protocols. It's useful to scan this to see what tasks protocols have been built for. This defines which RFC's are actual standards, as opposed to requests for comments.
rfc1010	Assigned Numbers. If you are working with TCP/IP, you will probably want a hardcopy of this as a reference. It's not very exciting to read. It lists all the offically defined well-known ports and lots of other things.
rfc1009	NSFnet gateway specifications. A good overview of IP routing and gateway technology.
rfc1001/2	netBIOS: networking for PC's
rfc973	update on domains
rfc959	FTP (file transfer)
rfc950	subnets
rfc937	POP2: protocol for reading mail on PC's
rfc894	how IP is to be put on Ethernet, see also rfc825
rfc882/3	domains (the database used to go from host names to Internet address and back -- also used to handle UUCP these days). See also rfc973
rfc854/5	telnet - protocol for remote logins
rfc826	ARP - protocol for finding out Ethernet addresses
rfc821/2	mail
rfc814	names and ports - general concepts behind well-known ports
rfc793	TCP
rfc792	ICMP
rfc791	IP
rfc768	UDP
rip.doc	details of the most commonly-used routing protocol
ien-116	old name server (still needed by several kinds of system)
ien-48	the Catenet model, general description of the philosophy behind TCP/IP

The following documents are somewhat more specialized.

rfc813	window and acknowledgement strategies in TCP
rfc815	datagram reassembly techniques
rfc816	fault isolation and resolution techniques
rfc817	modularity and efficiency in implementation
rfc879	the maximum segment size option in TCP
rfc896	congestion control
rfc827,888,904,975,985	EGP and related issues

To those of you who may be reading this document remotely instead of at Rutgers: The most important RFC's have been collected into a three-volume set, the DDN Protocol Handbook. It is available from the DDN Network Information Center, SRI International, 333 Ravenswood Avenue, Menlo Park, California 94025 (telephone: 800-235-3155). You should be able to get them via anonymous FTP from sri-nic.arpa. File names are:

```
RFC's:
  rfc:rfc-index.txt
  rfc:rfcxxx.txt
IEN's:
  ien:ien-index.txt
  ien:ien-xxx.txt
```

rip.doc is available by anonymous FTP from topaz.rutgers.edu, as /pub/tcp-ip-docs/rip.doc.

Sites with access to UUCP but not FTP may be able to retreive them via UUCP from UUCP host rutgers. The file names would be

```
RFC's:
  /topaz/pub/pub/tcp-ip-docs/rfc-index.txt
  /topaz/pub/pub/tcp-ip-docs/rfcxxx.txt
IEN's:
  /topaz/pub/pub/tcp-ip-docs/ien-index.txt
  /topaz/pub/pub/tcp-ip-docs/ien-xxx.txt
/topaz/pub/pub/tcp-ip-docs/rip.doc
```

Note that SRI-NIC has the entire set of RFC's and IEN's, but rutgers and topaz have only those specifically mentioned above.

Future Berkeley UNIX developments

Mike Karels
Computer Systems Research Group
Computer Science Division
University of California
Berkeley, CA 94720
USA
karels@Berkeley.EDU

Computer Systems Research Group Staff

- Head Honcho

  Domenico Ferrari

- Movers and shakers

  Mike Karels

  Kirk McKusick

  Keith Bostic

  Keith Sklower

- DEC Berkeley Division

  Jean Wood

  Marc Teitelbaum

- Distribution Office

  Pauline Schwartz

  Anne Hughes

4.3BSD Hardware Base

- VAX
  - 8600/8650, 780/785, 750, 730
  - MicroVAX II (3X00 recently)
  - 8200/8250 (BI bus)
  - *not* 85x0, 8700, 8800

- Tahoe
  - CCI Power 6/32, 6/32SX
  - Harris HCX-7 (HCX-9 soon)
  - Sperry 7000/40
  - ICL Clan 7

Recent Work:
4.3BSD Tahoe release

- Support for Tahoe hardware, VAX 8250
- New TCP algorithms
  (slow start, congestion control)
- Kernel memory allocator
- Disk labels
- Flexible filesystem limits

4.3BSD Kernel Memory Allocator

Criteria for a Kernel Memory Allocator

- Speed of allocation

  ‘‘These routines are not that fast, so they should not be used in very frequent operations (e.g. operations that happen more often than, say, once every few seconds).’’

- Effective utilization of memory

  ‘‘While the memory allocator distributed with 4.2BSD was more than twice as fast as the other memory allocators that were tested, it used twice as much memory to do the allocations.’’

4.3BSD Kernel Memory Allocator

- Like the 4.3BSD C library malloc, the kernel *malloc* uses a buddy system when doing allocations of a page size or less.
- Macros are available for frequent allocation paths.
- Allocations larger than page size use a slower but more memory-efficient allocator.
- Each page holds only one allocation size.
- Replaces many special-purpose allocators such as calloc, wmemall, zmemall, geteblk, and m_get.

Disk Pack Labels

- Labels on each disk (pack) contain
  - detailed geometry information (cylinder, track, sector layout; driver-specific information)
  - partition layout and usage
- stored in block 0 (shared with boot block)
- installed and modified with *disklabel* (8)
- used by

  kernel, bootstrap (autoconfig, drivers, error reporting)

  newfs (geometry; filesystem usage)

  fsck (alternate superblocks)

  bad144 (bad-sector table)
- Filesystem now understands irregular rotational layout (track skew, interleaving) (done by newfs)

Current CSRG projects

- routing protocols and implementations (routed, EGP and gated)
- Internet nameserver and extensions
- ISO/OSI Networking protocol support
- updates to system interface
- POSIX-compliant interface
- Generic Virtual Filesystem Interface Switch
  - in progress
- rearrangement of the UNIX filesystem
  - in progress

4.3BSD POSIX interface

- Terminal interface
  - POSIX interface derives from System V
  - 4.3 interface derived from Seventh Edition, with numerous additions
  - 4.4BSD will use POSIX interface with obvious extensions
  - prototype now running
  - utilities mostly converted to new interface
- Job control
  - POSIX job control derived from 4.2BSD, added security model
  - prototype now running
- Signals
- Group sets

Virtual Memory design

- Processes as regions
    - shared memory with semaphores
    - file mapping (private or shared)
    - device memory mapping
    - copy on write after fork and file mapping
    - lightweight processes

- Large, sparse address spaces
    - multilevel, paged data structures
    - swap image not preallocated

- Reasonable hardware independence, range of configurations

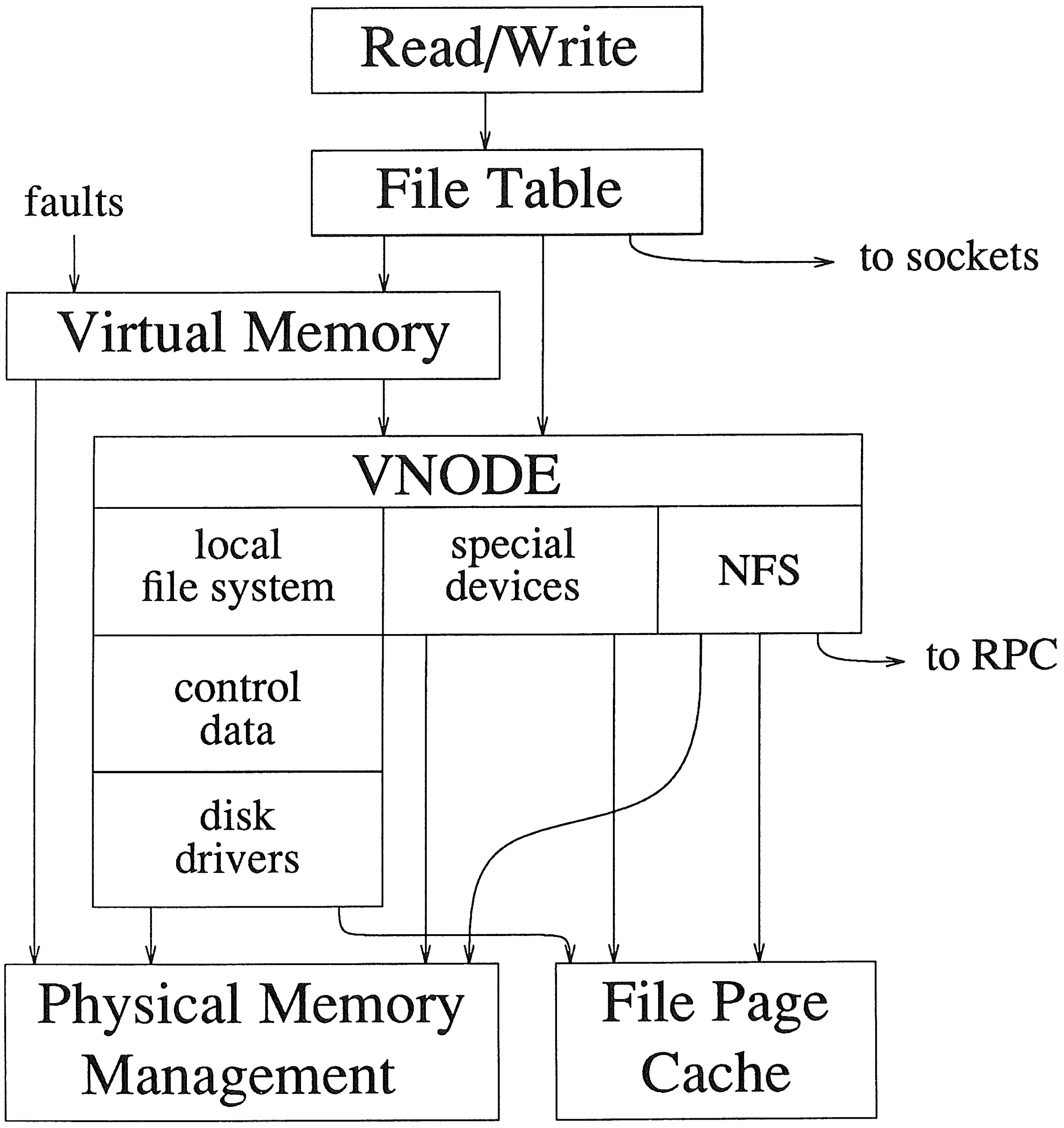
- Integration with kernel memory allocation, network, filesystem

Virtual Memory design

- User interface finalized
  - *mmap, munmap*
    mprotect, madvise, mincore; msync
    mset, mclear; msleep, mwakeup
  - *mlock* added
  - All naming of memory objects uses filesystem namespace; virtual-memory-resident filesystem for transient objects

- Interaction with filesystem
  - Filesystems own ``buffer cache'' pages
  - Virtual memory system owns private pages
  - Pages may be both mapped and ``cached'' (copy on write)
  - Size of buffer cache varies according to memory demand
  - Page fault service calls filesystem implementation to find/get fill-from-file data.

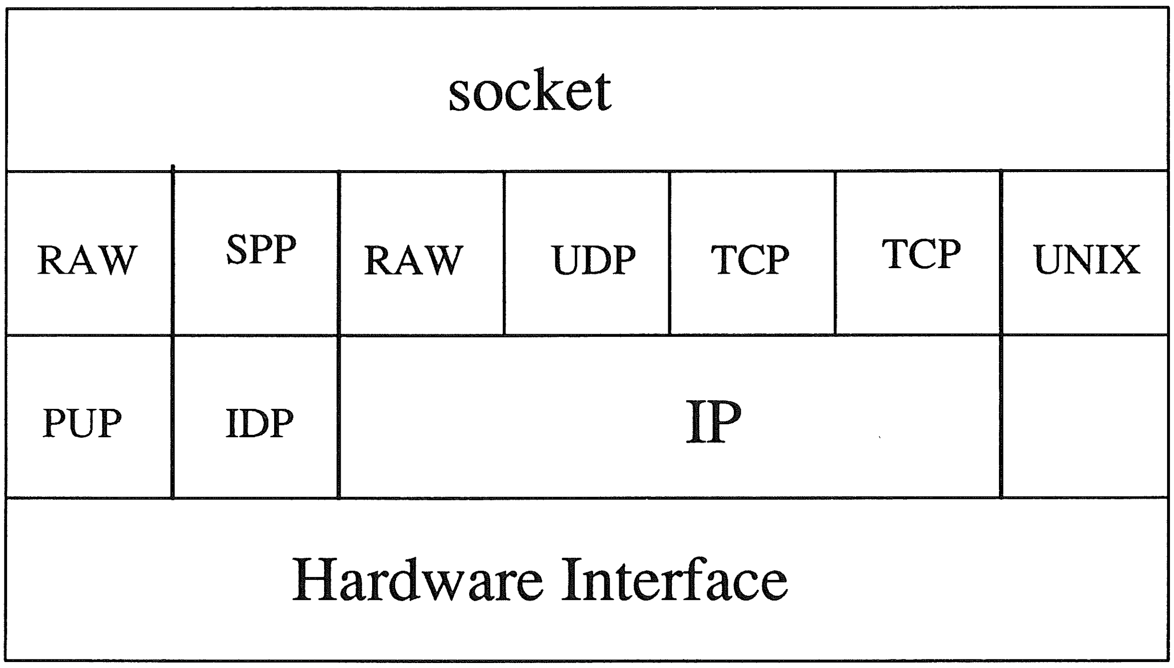
Kernel Structure



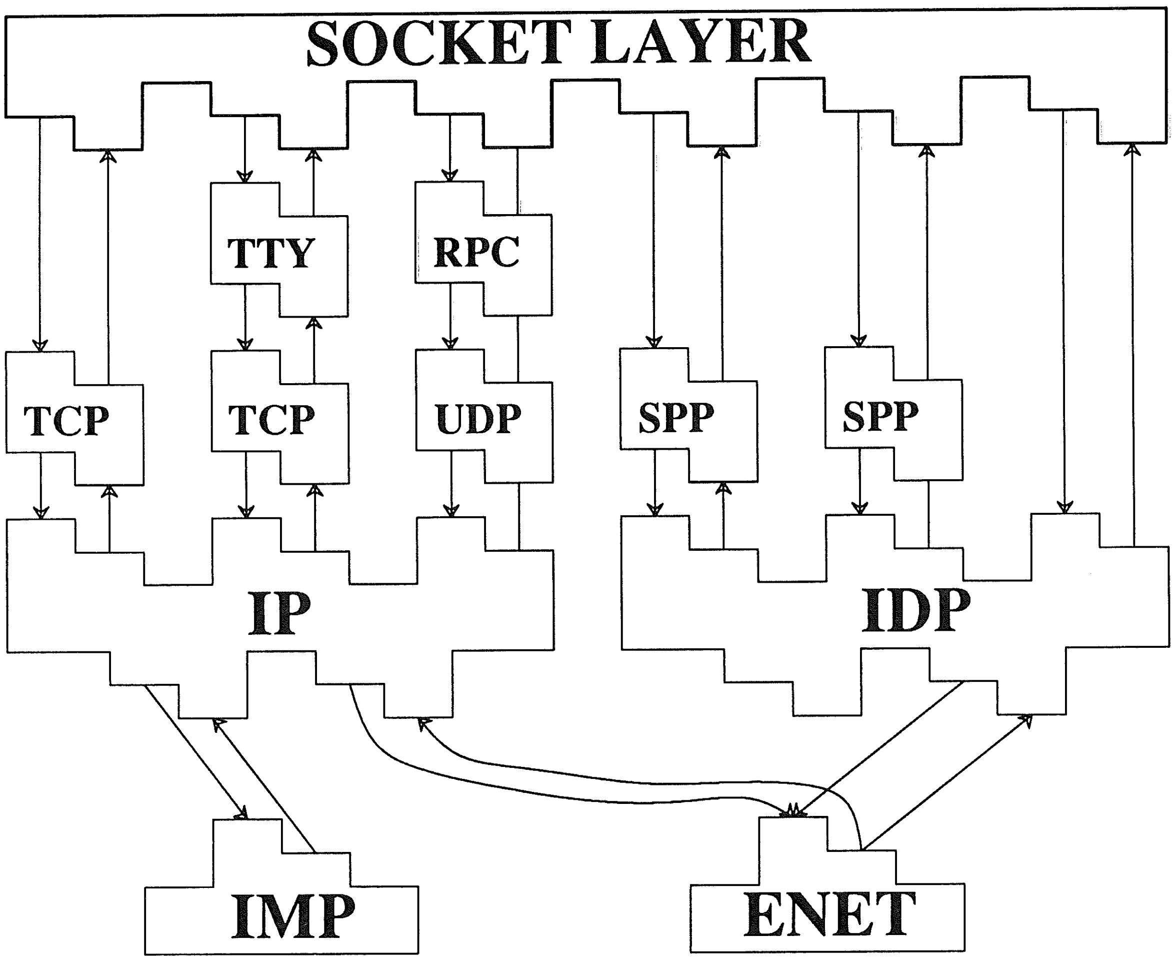
Protocol layering interfaces

- 4.2, 4.3 BSD
    - three layers, three interfaces
    - Protocols are top-level (TCP, UDP) or bottom-level (IP, IDP)
    - Terminal line disciplines use separate interface, buffering, control
- Eighth Edition Unix (System V.3)
    - uses ‘‘Stream’’ abstraction
    - stylized interaction allows easy stacking
    - protocols/line disciplines as coroutines
    - multiplexing is difficult
- 4.4 BSD
    - Unify stream/tty interface and protocol interfaces
    - kernel-level demultiplexing as in 4.3BSD
    - stackable processing modules above demultiplexing layers

4.2/4.3BSD Protocol Layering



New Protocol Layering



Licensing

- 3BSD, 4BSD, 4.1BSD, 4.2BSD, and 4.3BSD are based on AT&T 32/V UNIX
- Should the next BSD release incorporate later AT&T technology?
- Currently considering the use of System V, release 2 technology.

Kernel Internal Cleanups

- merge proc, user structures
    - dynamically allocated
    - always resident
- reduce global use of user structure
    - expunge u.u_error
- New sleep/wakeup mechanism
    - eliminate aborted system calls (longjmp)

;login:

The USENIX Association Newsletter

Volume 14, Number 1 January/February 1989

CONTENTS

The closing date for submissions for the next issue of *;login:* is February 24, 1989

If you have not paid your 1989 membership dues, this is your last issue of *;login:*. Use the form on the inside back cover (p. 51).

USENIX THE PROFESSIONAL AND TECHNICAL UNIX® ASSOCIATION

Call for Papers

Workshop on Software Management

New Orleans Hilton and Towers
New Orleans, LA

April 3-4, 1989

David Tilbrook and Barry Shein will be chairing a workshop in New Orleans, LA on Monday and Tuesday, April 3-4, 1989. The workshop will concern the management and processing of source, and the discipline of managing, maintaining, and distributing software. The ultimate objective of software management is the unremarkable and painless installation of software and its subsequent upgrades at a remote site. The objective of the workshop is to present, discuss, and increase awareness of the issues involved with software management, in order to improve and facilitate the distribution and sharing of source throughout the UNIX community. Possible topics include:

Release engineering
Configuration management
Installation tools and techniques
Construction tools and techniques
Source code control systems
Testing

The workshop will include full length papers, short presentations, and a panel discussion on tools (e.g., is PCTE a good or viable idea?).

Among the speakers already scheduled are Vic Stenning (keynote), Steve Bourne, Andrew Hume, Kirk McKusick, and Evi Nemeth.

Abstracts of 350-700 words in PostScript or *troff* format should be submitted to *software@usenix*, by **January 25, 1989.** Full papers will be required by **March 2, 1989.** Authors will be notified by **Febuary 6, 1989** or at the San Diego Conference.

Call for Papers
Workshop on UNIX Transaction Processing

Pittsburgh Hilton Hotel
Pittsburgh, PA

May 1-2, 1989

It is expected that the UNIX System will play an increasingly important role in hosting production transaction processing systems. This first transaction processing workshop will explore existing technology applicable to UNIX-based transaction processing, and hopefully generate technical discussion on future requirements. The intent is to have short papers and presentations which include (but are not limited to) the following topics:

Transaction Integrity
Two-Phase Commit
Distributed Transactions
Client-Server Transaction models
Transaction queing and scheduling
Data Entry Systems
Transaction Benchmarking
Transaction system performance modelling
Operating System Support for Transaction systems

The workshop will focus on short papers and presentations. Please send electronically or on paper a one to two page single-spaced summary describing your paper or presentation to Doug Kevorkian by **February 1, 1989**. All submissions will be acknowledged, and authors will be notified of acceptance by March 15, 1989.

For further details about the workshop, contact the program chair:

Doug Kevorkian
AT&T Bell Laboratories
Room 5-340
190 River Road
Summit, New Jersey 07901

(201) 522-5086 (voice)
(201) 522-6621 (FAX)
attunix!dek

Call for Papers Summer 1989 USENIX Conference

Baltimore, Maryland June 12-16, 1989

Papers in all areas of UNIX-related research and development are solicited for formal review for the technical program of the 1989 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:

Performance:
- Kernel enhancements
- Compute and file servers
- Scaling issues resulting from more MIPS

File systems: CD-ROM, WORM, network, archival
Networks: WAN, LAN, UUCP, OSI, distributed services
User interfaces
High reliability/availability, fault tolerance
Heterogeneous environments: mainframes, DOS/UNIX migration
Media: graphics, video, audio, art, education
System/network administration and security
Trends:
- Lightweight processes
- Neural networks
- Object-oriented extensions

All submissions should describe new and interesting work. Like recent USENIX conferences, the Baltimore conference is requiring the submission of full papers rather than extended abstracts. The 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. All final papers must be submitted in a format suitable for camera-ready copy. For authors that do not have access to a suitable output device, facilities will be provided.

An abstract should be submitted as soon as possible. Full details and requirements will be supplied to prospective authors. Copies of the full manuscript should be submitted by ordinary and electronic mail to the Program Chair. Electronic submissions are recommended; *troff -ms* if possible.

Four copies and one electronic copy of each submitted paper should be received by **February 8, 1989**. Papers not received by this date will not be considered. 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 made by March 13, 1989, and final camera-ready papers will be due by **April 7, 1989**.

Neil Groundwater
Baltimore USENIX Technical Program
Sun Microsystems, Inc.
8219 Leesburg Pike #700
Vienna, Virginia 22180
(703) 883-1221

Abstracts, submissions, and questions:

usenet: {ucbvax,decvax,decwrl,seismo}!sun!balt-usenix
internet: balt-usenix@sun.com

Enhancing the 4.3 BSD UNIX Serial Line Interface

Alfredo Almada and David H. Williams

Department of Computer Science
The University of Texas at El Paso
El Paso, Texas 79968

ABSTRACT

This paper describes simple modifications to the 4.3 BSD UNIX[†] serial line interface that allow serial lines to be individually customized according to the devices to which they are connected. This allows nonstandard terminals such as graphics displays, and nongetty devices such as plotters to invoke hardware or software flow control, and to achieve the proper amount of I/O processing in a manner that is transparent to the user. Previously these lines suffered from a lack of I/O processing such as the inability to invoke software handshaking, and over processing such as the generation of extra characters which corrupted commands for graphics displays. The solution consisted of setting the proper combination of parameters in the provided databases *gettytab* and *printcap*, and the creation of a new database, *rawtab*, that was employed to initialize ports not covered by the other databases. In addition, minor modifications were made to the program *getty*. A complete description of the serial line interface and its internals is provided as reference for possible future updates.

1. Introduction

This work results from several years of operational experience with a DEC VAX[‡] computer running Berkeley 4.2 and later 4.3 UNIX. During this time we found that the serial line communications were quite satisfactory for standard alphanumeric terminals, and conversely, that communications for nonstandard devices were unsatisfactory and required hacks for a semblance of proper operation. Nonstandard devices in our case consisted of "nongetty" devices which did not require a login (i.e. plotters), and graphics displays which might be used for text, but mostly were employed for graphics. Most larger installations have these types of devices connected to serial lines.

Hacks included setting the baud rate manually on "nongetty" devices that did not operate at the default baud rate, ignoring flow control by using the raw interface, or working around character mappings when a terminal was under the control of getty. In some cases, application software was written to manually set the line parameters each time the program was executed and the device was opened.

In our opinion, these processes violated the tenet of UNIX that all I/O activity should be identical to the user, regardless of the file (device) that is being accessed. Furthermore, it is the duty of the operating system to make this process transparent to the user. Output redirected to nonlogged in alphanumeric terminals should exhibit <cr> <lf> mapping; output only devices should exhibit proper flow control and baud rate settings; graphics and other nonalphanumeric terminals should have certain special characters enabled (e.g. xon, xoff) and others disabled to prevent "hits" with internal commands within the terminal.

The purpose of this paper then is twofold: to describe in a detailed manner how the 4.3 BSD UNIX serial interface works, and to describe simple modifications that have been made to the operating system in order to enhance the serial line interface with nonstandard devices. The paper illustrates how the terminal interface is handled internally by the kernel, and also discusses the different system calls available to the user to

‡ DEC & VAX are Trademarks of Digital Equipment Corporation.

† 4.3 BSD UNIX is copyrighted by the Regents of the University of California at Berkeley.

properly communicate through terminal lines. Although it describes most of the terminal driver capabilities, the reader should refer to the existing documentation (i.e. manuals) for a complete description of what is available. The specific processes apply only to 4.3 BSD UNIX executing on DEC VAX computers, however, many of the overall principles should be applicable to other versions of UNIX as well.

2. File I/O

To the UNIX user, all file I/O activity should look the same without regard to what kind of file is being employed. That is, the same system calls and processes such as redirection are invoked to access a regular file, a terminal line, a tape drive, and so on. This section deals with how the operating system makes these actions transparent to the user, and provides a description of the kernel to device driver interface and also a discussion of the system calls related to file management.

2.1 Internal Representation of Files

The kernel handles all file I/O internally through the use of inodes. Inodes exist on disk, and the kernel reads them into in-core inodes when they become active (the first instance of the file is opened) to manipulate them. The inode contains information such as owner and group identifiers, access permissions, access times, size, and type. The in-core copy of the inode contains additional information such as whether it is locked, whether someone is waiting for it to become unlocked, whether it has been modified, the inode number, and the reference count (the inode contains more information, however, only information relevant to the scope of this paper has been presented) [1].

Depending on the type of the file, the inode contains information that serves the purpose of that file. For regular files and directories, the inode contains the disk block addresses in which the data of the file are located. In the directory case, the data are the names and inode numbers of the files in the directory. For character and block device special files, the inode contains the major and minor device numbers, which uniquely identify a device. The major number distinguishes among the different types of devices, such as terminals, disk drives, tape drives, etc., and the minor number differentiates among several devices of the same type. (Two other types of inodes, symbolic links and sockets, are not discussed in this paper.)

Eventually, the last instance of the file will be closed as indicated by a reference count of zero. This will cause the inode to be written back out to disk and possibly deallocated from the in-core table, in the event that another disk inode is waiting for a free spot.

2.2 Kernel I/O Interface with Device Special Files

There are two kinds of interfaces with which the kernel communicates with external devices [2]. They are the block and character interfaces. The block interface provides a buffering mechanism, for which the algorithms of the buffer cache are invoked. The character interface is a faster raw interface which bypasses the buffer cache. The two interfaces are implemented through the use of the block device switch table *(bdevsw)* and the character device switch table *(cdevsw)* respectively. From these tables the kernel takes the entry points to the specific driver routines to be used when invoked by the different system calls.

The major number of a device, taken from the inode, is used as an index into the bdevsw or to the cdevsw depending on the type of the device. The minor number is passed to the selected routine so that it can identify the particular device. Both interfaces contain entry points for the open and close procedures. The mount and umount system calls also invoke the device open and close procedures for block devices. The read, write, and ioctl system calls for the character interface also get their entry points to the driver from the character device switch table. However, read and write system calls of block devices and of files that are on mounted file systems invoke the algorithms of the buffer cache, which invoke the device strategy procedure. This is the entry point contained in the block device switch table. The routine nulldev is used when there is no need to perform a particular driver function. However, the routine nodev is used when it should be considered an error to try to perform that driver function, such as if that device was not configured.

The cdevsw also provides entry points to routines for other more device specific tasks, such as a stop procedure for terminal multiplexers to stop transmitting on a given line, and reset routines for those devices that need to do so. There are other fields in these tables that are not relevant to this paper and and therefore are not discussed. Figure 2.1 illustrates the format of the bdevsw and cdevsw tables, and the following sections describe system calls which utilize their information.

2.3 Related System Calls

2.3.1 open

A file must be opened before it can be manipulated. The user opens a file with the *open* system call. The syntax is as follows:

i = open(path, flags, mode)

where path is the pathname of the file; flags is a set of actions to be taken when opening the file such as whether the file should be opened for reading, writing, whether it should be created, truncated, etc.; mode specifies the mode of the file if it is to be created; and i is the file descriptor returned by the system call [3].

The open system call allocates a file structure from the system-wide file table, and a file descriptor from the per-process u_ofile table, which is located in the u area of the process. The allocated u_ofile entry points to the allocated file structure [4]. The pathname is then parsed into an inode *(namei)*; if this is the first instance of the file, the inode is read from disk into an in-core inode and the reference count is initialized to 1; otherwise, there is already an in-core copy of the inode and its reference count is incremented (the allocation of inodes, and the mapping from disk inodes to in-core inodes is beyond the scope of this paper) [1]. In any case, a pointer to the in-core inode is obtained and stored in the file structure. The file structure has, in addition, other fields that are initialized during the opening process. There is a set of flags saved after the open that indicates whether the file was opened for reading, writing, or appending after each write. There is also a type field which indicates that the referenced object is an actual "file" (DTYPE_INODE) and not a communications endpoint (socket). In addition, a structure within the file structure is initialized with the entry points of the general I/O routines to be invoked when the user references the file. These are *read/write, ioctl, select,* and *close.* The reference count is initialized to 1. This is a different reference count not to be confused with the inode reference count. The reference count in the file

BLOCK DEVICE SWITCH TABLE

Index	Open	Close	Strategy	dump
0	hpopen	hpclose	hpstrategy	hpdump
1	upopen	nulldev	upstrategy	updump
2	rkopen	nulldev	rkstrategy	rkdump
3	udopen	nulldev	udstrategy	uddump
4	tsopen	tsclose	tsstrategy	tsdump

CHARACTER DEVICE SWITCH TABLE

Index	Open	Close	Read	Write	Ioctl	stop
0	cnopen	cnclose	cnread	cnwrite	ttioctl	nulldev
1	dmfopen	dmfclose	dmfread	dmfwrite	dmfioctl	dmfstop
2	udopen	nulldev	udread	udwrite	nodev	nodev
3	dzopen	dzclose	dzread	dzwrite	dzioctl	dzstop
4	tsopen	tsclose	tsread	tswrite	tsioctl	nodev

Figure 2.1: Sample of bdevsw and cdevsw

structure keeps track of how many descriptors are sharing the same instance of the file, while the inode reference count keeps track of how many instances of the file are open. Finally, the offset which determines where the next read or write will take place within the file is initialized to 0.

In the case of character and block device special files, the kernel invokes the specific open procedures to validate and initialize the driver private data structures before returning to the user. The file descriptor, which is the smallest available integer used as an index into the u_ofile table, is returned to the user. The user needs only this file descriptor to reference the file.

There is an instance of a u_ofile descriptor and a file structure entry allocated for each open call, however, there is only one inode allocated per active file in the system. See figure 2.2 for further illustration. At initialization, the kernel allocates space for its data structures. The structures relevant to file management are the process table (more specifically the u_ofile table inside the u area of the process), the file table and the inode table. The size of these tables generally depend on the maximum number of users specified at configuration time.

2.3.2 Dup and Fork System Calls

There are two other system calls besides open that allocate a file descriptor in the u_ofile table of the process. However, in contrast to the open system call, they do not allocate a file structure but instead share one that already exists. The first of these two calls is the *dup* system call. The dup call is invoked as

```
newfd = dup(i)
```

where i is a previously allocated descriptor [3]. It causes the lowest numbered descriptor available from the u_ofile table to be allocated, and the file structure being pointed to by the ith entry to now be shared between the two descriptors i and newfd. Figure 2.3 illustrates the effect of a dup call. The other call is the *fork* system call. During a fork, the u_ofile table of the parent process is inherited by the child process. As a result, the child process now shares with its parent all the file structures that had been allocated for the parent process. See figure 2.4 for illustration.

These two system calls cause the reference count in the particular file structure(s) to be incremented. As will be seen later, the close system call checks and decrements this reference count and calls the device closing procedures only when this count reaches zero.

2.3.3 Read/Write

The *read* and *write* system calls are implemented in very much the same way. In fact, the same internal kernel routines are used for both, and a flag set by the user callable read and write routines, determines which of the two operations is to be performed. The syntax for these two system calls is

```
i = read(fd, buf, count)
i = write(fd, buf, count)
```

where fd is a previously allocated descriptor from one of the system calls described above, buf is the address where data is to be stored (read) or taken from (write), count is the number of characters to be transferred, and i specifies the number of characters actually transferred [3]. The kernel sets up this address and character count in the uio (user I/O) structure, in addition to some other fields that will be explained below.

As described in the open procedure, the file descriptor specified by the user is used as an index into the u_ofile table of the process to obtain the pointer to the corresponding file structure. The flags that were saved in the file structure after the open are used to check that the intended operation (read/write) corresponds to what the file was opened for. The offset that indicates where this read/write should start is also taken from the file structure and stored in the uio structure. If the file is a regular file opened in write/append mode, the offset is set to the size of the file, taken from the inode. An additional flag set in the uio structure indicates to the kernel that it should transfer the data from kernel to user space in the case of a read, or from user to kernel space in the event of a write. Once the uio structure is properly initialized, the inode read/write routine is invoked with the file table entry, the read/write flag, and the uio structure as arguments. On return, the offset is updated

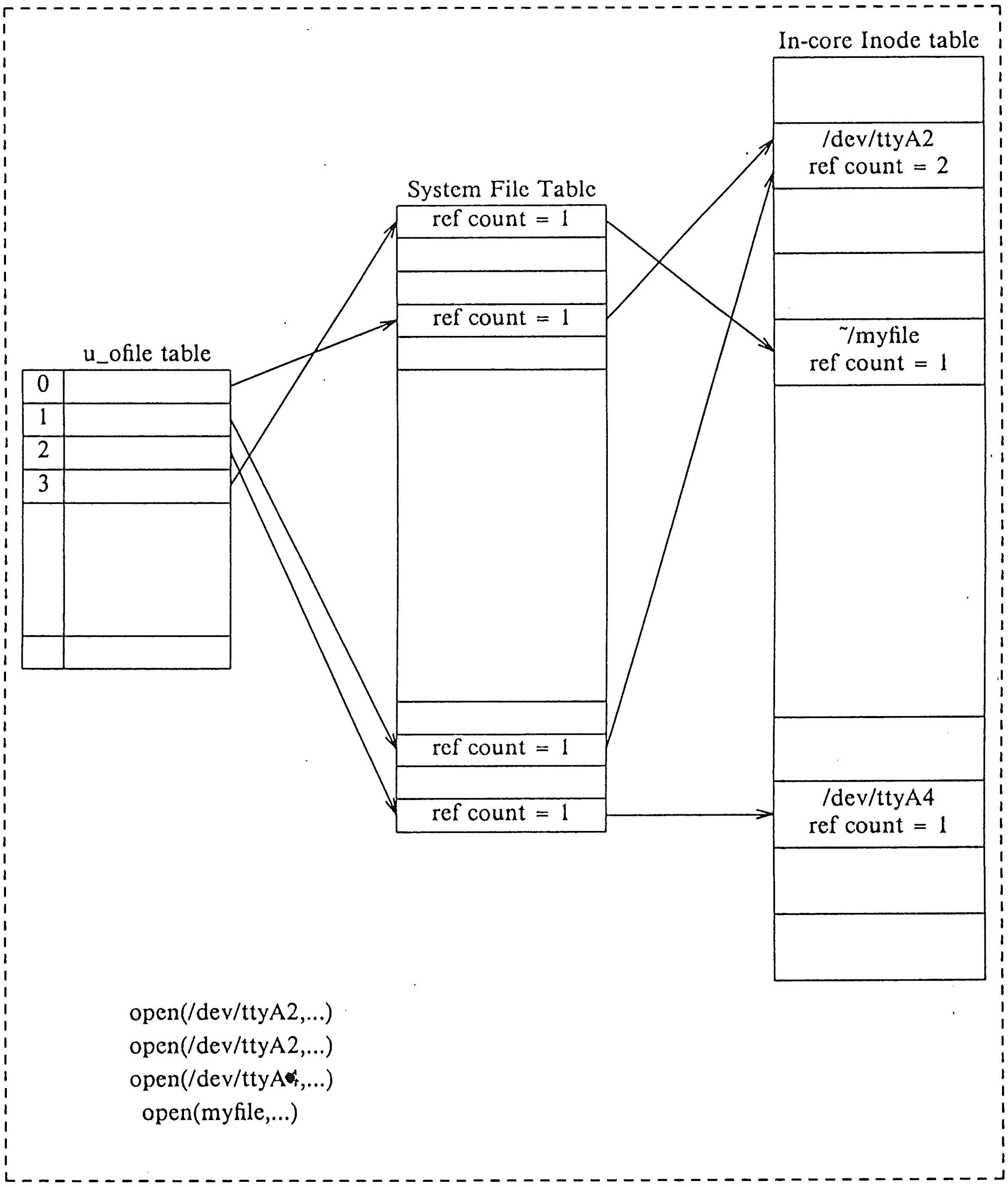


Figure 2.2: Effect of the open system call on file tables

and the actual number of characters read/written is returned to the user.

The inode read/write routine checks the type of the inode and performs the appropriate task. In the case of block devices and files that are on mounted file systems (regular files, directories, and symbolic links) the routine invokes the algorithms of the buffer cache, which in turn invoke the device strategy procedure (bdevsw) [2]. The other type of inodes are character device special files. For this type of inodes, the major and minor numbers are obtained from the inode itself. The major number indexes the cdevsw to pick the device driver entry point for a read or write operation. The minor number and the uio structure are passed as arguments to this routine. See figure 2.5 for illustration.

When a read occurs, the inode is marked as accessed so that its access time is modified. When a write occurs, the inode is marked as updated meaning that the file has been modified, and as changed meaning the the inode itself has changed. Updated and changed are two different states. When a file is updated, the inode is consequently modified. However, the inode can be modified without the file being changed, such as when changing the ownership of the file.

2.3.4 Ioctl

The *ioctl* system call is used to customize communication parameters for devices. It is mostly intended for character devices (in particular, terminal lines) and communications endpoint types of files (sockets). Nevertheless, some limited requests are allowed on regular files and directories. Its syntax is as follows:

ioctl(fd, request, argp)

where fd is again a previously allocated descriptor, request is the action to be performed, and argp is a pointer to the parameter list [3]. Request is an unsigned quantity four bytes long, each of which contains specific information about the request. The high order byte indicates whether the parameters pointed to by argp are *in* and/or *out* parameters, or neither. The next byte contains the size in bytes of the parameter list pointed to by argp; therefore a maximum of 127 bytes is allowed. The next byte contains the ascii code of a literal of the set { t, f, s, r, i } that identifies the class of the "file" on which this ioctl command is going to be performed; t is used for character devices (i.e. terminals), f is for regular files or directories, and s, r, and i are used for sockets. The low order byte contains a unique integer id within the class to identify the particular command.

The ioctl system call funnels through the cdevsw in the same way that the previously discussed routines do. Because of its particular correlation with terminal lines, this relationship will be deferred until the section describing the terminal I/O interface.

2.3.5 Close

Eventually, the user files are closed. Either this operation is performed by the user, or if a process still has open descriptors when it exits, all are closed automatically during the exit system call. The *close* system call is provided for closing files and is as follows:

close(fd)

where fd is a previously allocated descriptor [3]. The close system call deallocates the fdth entry from the u_ofile table of the process, and decrements the reference count of the corresponding file table entry. If the reference count reaches zero, the closing procedures for this instance of the file are called. A reference count of zero in a file table entry means it is now free and can be reallocated to another descriptor. If the reference count is not zero the close system call returns immediately.

If the reference count in the file table entry indeed reached zero, it means there is now one less instance of the file open; thus the inode reference count is decremented to reflect the change. If the inode reference count is not zero, the system call returns to the user. On the other hand if it does drop to zero, the inode is written out to disk and possibly deallocated from the in-core inode table (if another disk inode is waiting for a free spot). For block and character devices the device closing procedures are invoked only at this point.

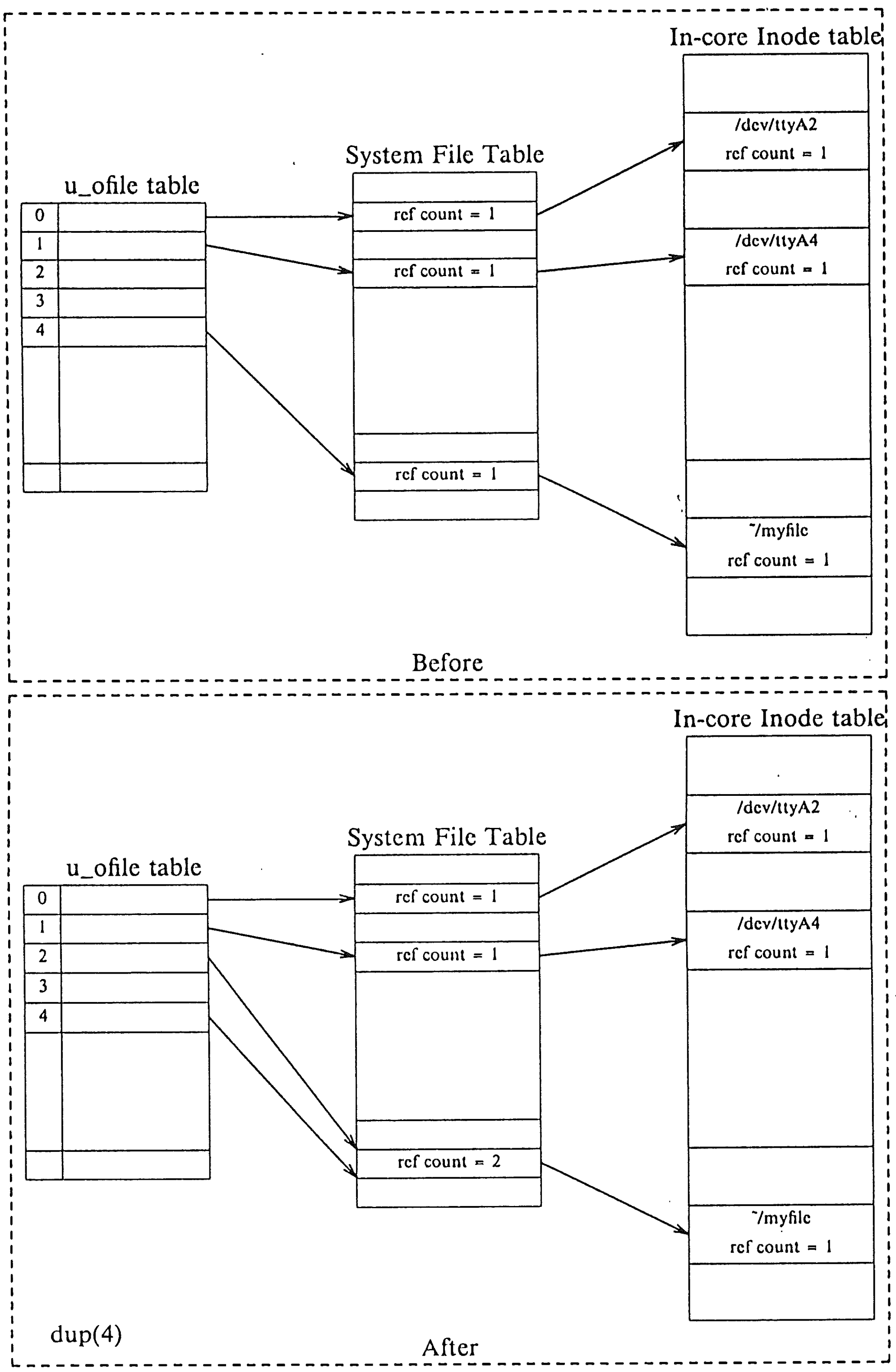


Figure 2.3: Effect of the dup system call on file tables

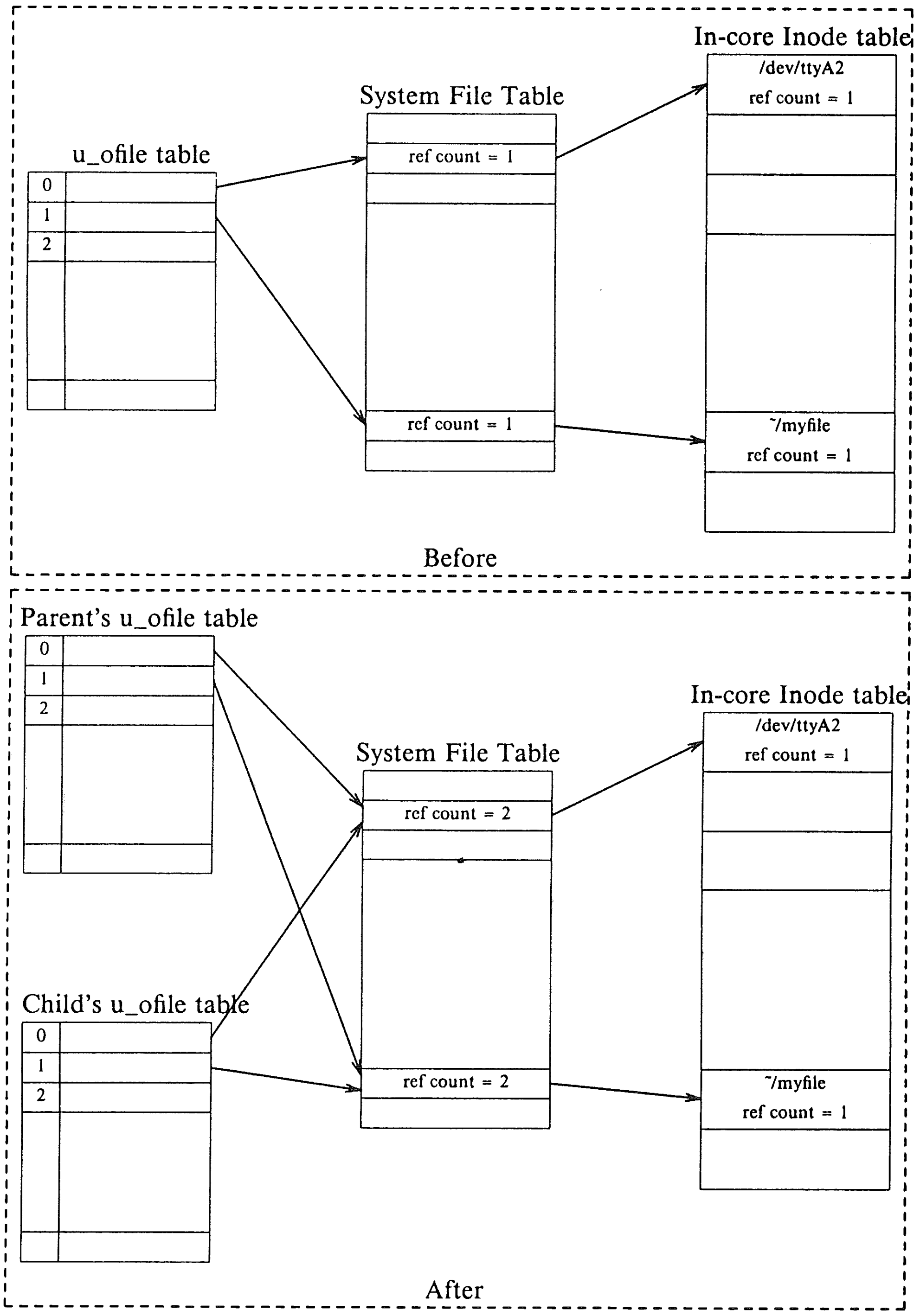


Figure 2.4: Effect of the fork system call on file tables

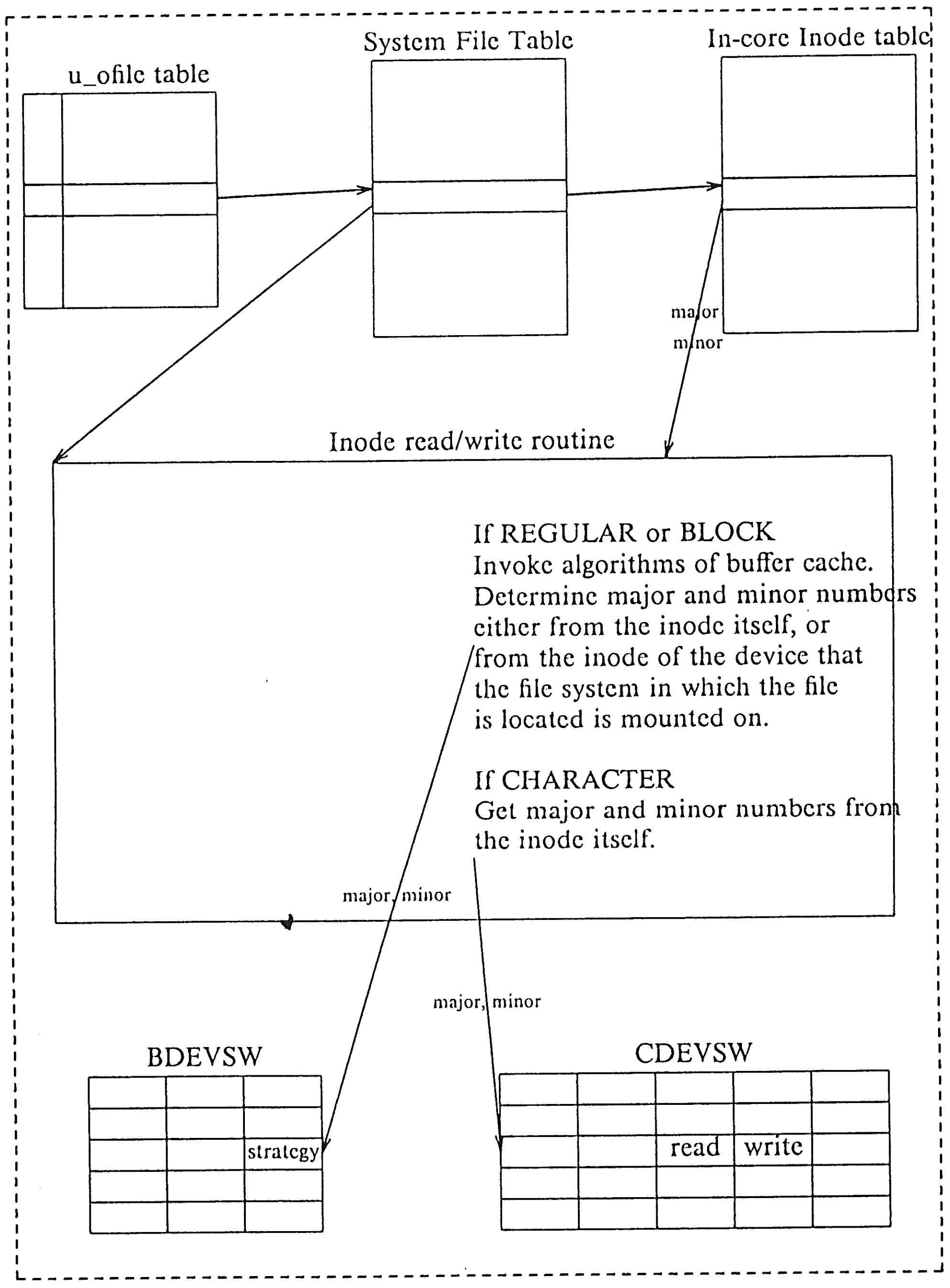

u_ofile table
System File Table
In-core Inode table
major
minor
Inode read/write routine
If REGULAR or BLOCK
Invoke algorithms of buffer cache.
Determine major and minor numbers
either from the inode itself, or
from the inode of the device that
the file system in which the file
is located is mounted on.
If CHARACTER
Get major and minor numbers from
the inode itself.
major, minor
major, minor
BDEVSW
strategy
CDEVSW
read
write

Figure 2.5: Illustration of the read and write procedures

3. Terminal I/O Interface

Terminal lines are a special case of character devices. In reality, terminal lines are usually controlled by terminal multiplexers, each of which controls several lines. The terminal I/O interface is controlled by the terminal multiplexer's driver, which in turn invokes the line discipline handling procedures for the different terminal lines. Line disciplines control the entire operation of terminal lines. They take care of opening and initializing the terminal state, performing all terminal settings, and providing a buffering mechanism appropriate for slow asynchronous communication lines, such as terminals. They also process input characters passed from the terminal multiplexer's interrupt service routine, and perform specific actions dictated by the different control characters. This part refers exclusively to the 4.3 BSD terminal line interface; however, many of the principles and examples still apply to other UNIX versions.

3.1 Terminal Multiplexers

A terminal multiplexer is the actual hardware device that controls the operation of terminal lines. When terminal multiplexers are configured into the system, the address of their control and status register and the names of the interrupt routines to call are specified in the configuration file [5]. The config program takes this information and produces a set of machine dependent routines to be invoked when the different types of interrupts occur [6]. These routines of course need to be compiled into the kernel. For a DEC VAX dmf device, the driver examines its configuration and adjusts the interrupt vectors during autoconfiguration.

The terminal multiplexer's driver entry points are located in the cdevsw entry corresponding to the major number of the device. These devices contain special registers for the hardware communications parameters of the different terminal lines. These include speed, parity, character length (8 or 7 bits) and modem control bits. There are other software communications parameters, such as control characters, terminal modes, etc., that are kept in the particular tty table entry of the terminal line. This part will refer to DEC VAX dmf 32 terminal multiplexers, each of which controls eight terminal lines; however, many of the principles also apply for other terminal multiplexers.

3.2 Line Disciplines

While terminal multiplexer drivers manage the hardware communication parameters like speed, character length, interrupt bits, and so on, they also invoke the specific line discipline procedures for the different terminal lines. Line disciplines control the operation of terminal lines. Upon opening, the line discipline open procedure establishes a control process group and a control terminal for distribution of signals. Line disciplines also handle all terminal I/O providing a buffering mechanism through the use of *clists* (character lists). The line discipline interfaces work in the same way that the character or block device interface work. The line discipline number is used as an index into the line switch table (linesw) from which the appropriate entry points to the tty driver are obtained. Refer to figure 3.1 for illustration.

There are two line disciplines available with the terminal interface. The first one is the old line discipline which is used with the Bourne shell. The second one is the new line discipline which has some additional job control features and must be used when using the C shell. Other disciplines may exist for special purposes, such as communications lines for network connections.

3.3 Clists

Clists provide a buffering mechanism for slow, asynchronous communication lines. A clist contains the number of characters in the list, and pointers to the first and last characters in the list. A clist is formed by a linked list of *cblocks*. A cblock has two fields, one of which is a pointer to the next cblock in the list, and the other one is the character array containing the characters in the clist. At initialization, the kernel allocates space for a number of cblocks and initializes the freelist to contain all of these cblocks. As input is received from the terminals, new cblocks may be allocated to the particular input clist of a terminal. As characters are read from the clist and given to the reading process(es), empty cblocks are returned to the free list. Similarly, when a process

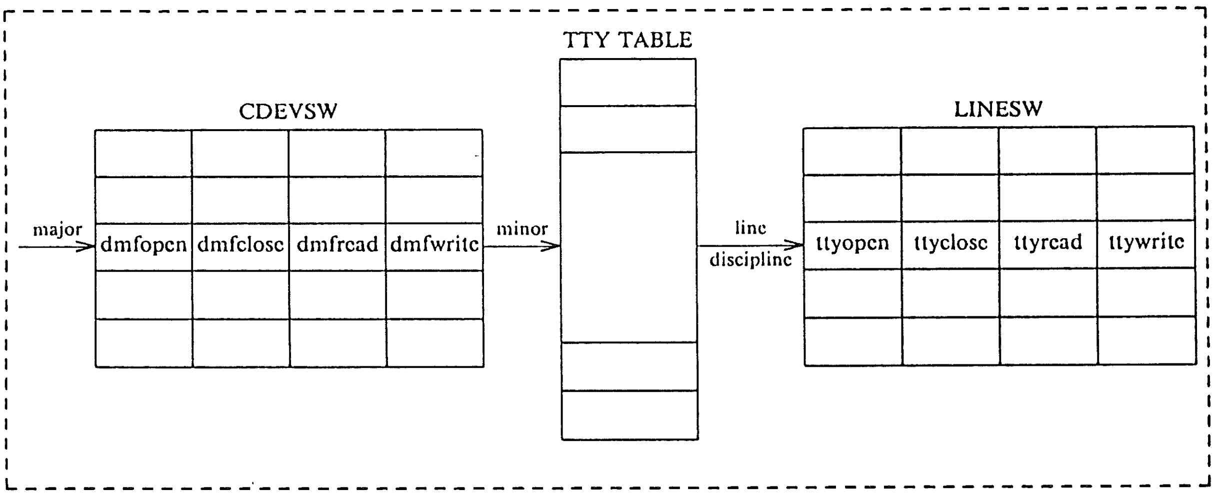


Figure 3.1: Illustration of the line discipline interface

writes to a terminal, new cblocks may be allocated for that terminal's output clist. As characters are transmitted from the output clist to the device's transmit buffer, empty cblocks are returned to the freelist. Figure 3.2 illustrates the process.

Every tty table entry has two input clists and one output clist associated with the particular terminal line [7]. The two input clists are called the *raw* queue and the *canonicalized* queue. Input characters are placed in the raw queue and transferred directly to the reading process as soon as they are input when in cbreak or raw mode (see below).. When in crmod, the raw queue is manipulated by the line editing functions. When any of the line terminating characters is recognized, the "updated" raw queue is transferred to the canonicalized queue, which is then given to the process. Obviously, the output queue is used for characters being output to the terminal.

3.4 Terminal Modes

Terminal I/O behavior is controlled by the terminal line mode. The following are the three different modes in which a terminal may bé operating:

crmod: This is the default mode. In this mode lines of input are collected and edited before making the line available to the reading process. The end of the line is recognized when either a <cr>, <nl>, EOT, or t_brkc (normally undefined) is entered. <cr> and <nl> are made synonymous in this mode and mapped to <cr><nl> on output. All driver functions, such as input editing, interrupt generation, output processing (such as delay generation and tab expansion), flow control, etc., are performed in this mode.

cbreak: In this mode charactersare made available to the user as they are typed. Therefore, no input editing facilities are performed. Flow control, literal-next, interrupt processing and output processing are still done.

raw: In this mode, 8-bit characters are placed in the input/output queue without being processed. None of the control and other special characters have any meaning whatsoever in this mode. On input, characters are made available to the reading process as soon as they are entered from the keyboard.

A fourth mode, *tandem*, can also be used in conjunction the above modes. In *tandem* mode, the system generates a stop character whenever the input queue reaches its high water mark, and a start character whenever the input queue empties to its low water mark. This mode is useful primarily for the communication between two CPUs and therefore will not be discussed further.

The driver recognizes the mode of the terminal line by checking the corresponding

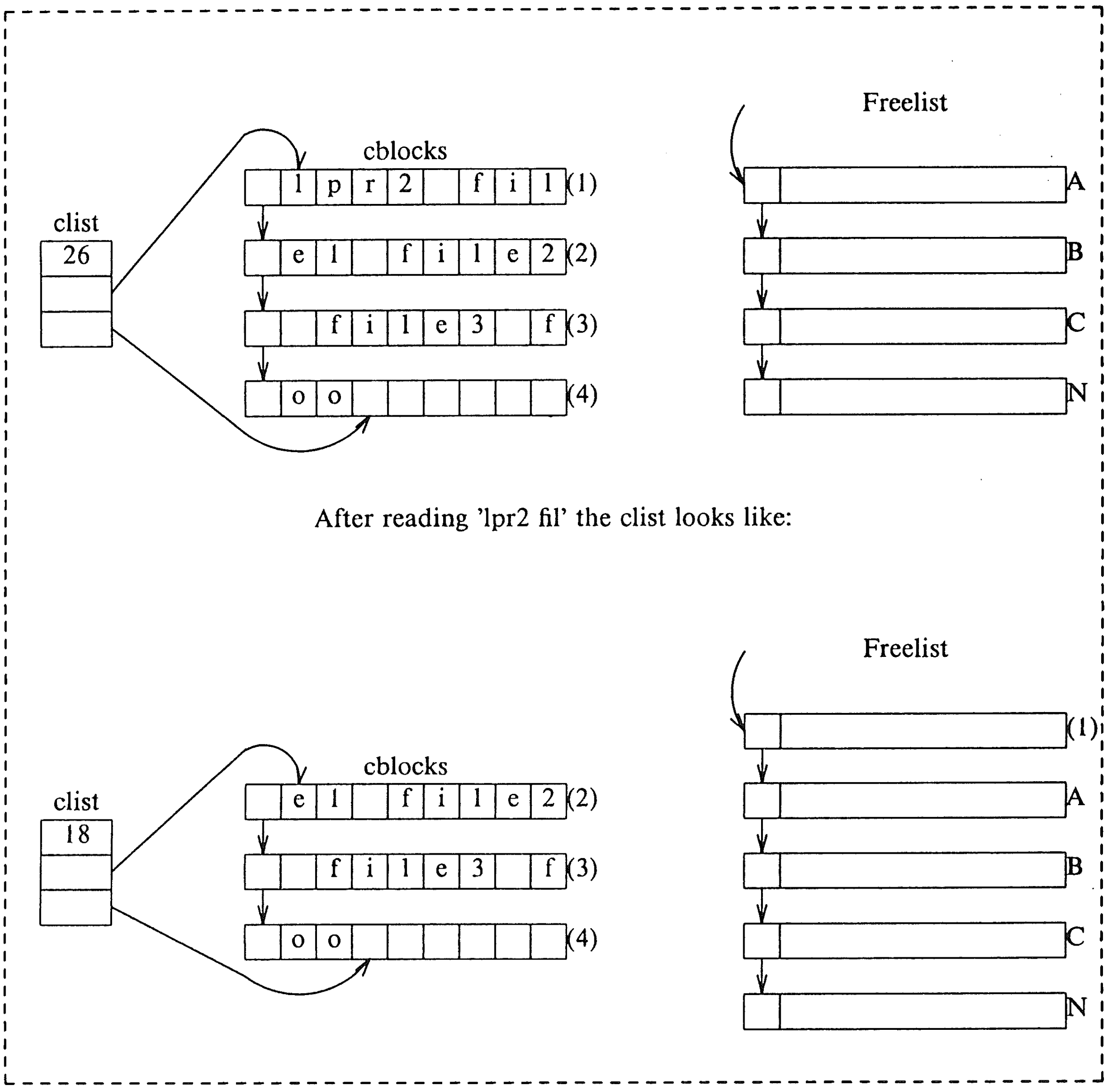


Figure 3.2: Reading characters from a clist

bit in the flags field of the particular tty table entry (refer to ioctl documentation [3] for additional information). If the raw bit is set, the other two bits are meaningless. That is, the terminal operates in raw mode as explained above. However, crmod and cbreak can be used alone or combined. If only one of the crmod or the cbreak bits is set, the terminal behaves as mentioned above. On the other hand, if both bits are set, the terminal operates as in cbreak mode, with the exception that the <cr> and <nl> characters are still mapped to <cr><lf> on output. If none of the three bits is set the terminal operates in crmod. There are other flags that alter the way the driver processes certain characters, some of which will be discussed in later sections. For a complete discussion of these flags and of the special control characters, the reader should refer to the ioctl and tty documentation [5].

3.5 Opening a Terminal

When a device is opened, the specific device open procedures are invoked as the last step in the open system call. For a terminal line, the terminal multiplexer open procedure is invoked through the cdevsw. The minor number is passed as an argument to this routine to identify the particular terminal line. This number serves also to identify the particular multiplexer and is used as an index into its tty table. The existence of the multiplexer is checked and the tty state is initialized to the default settings. This includes initialization of both the tty entry and the terminal multiplexer's registers. The default settings are 9600 baud, 7-bit character length and parity enabled (either). These are the actual hardware parameters kept in the terminal multiplexer's registers. There are also software parameters such as the control characters (start/stop, interrupt, quit, etc.), all of which are initialized to their default values and stored in the tty entry. These parameters, as well as most other terminal settings, can be customized using the ioctl system call.

The open procedure waits for a carrier signal and then calls the line discipline specific open procedure. The opening process will not wait for a carrier signal if the line was not configured to support full modem control, as in the case of hardwired lines. This is indicated by the lower byte of flags in the configuration file. If a bit is turned on, that line does not support full modem control. For a dmf device, the upper six lines should be configured in this way since the dmf itself does not support full modem control for those lines.

The main role of the line discipline open procedure is to establish a process group and a control terminal for distribution of signals. The opening process (usually getty) is the control process, and the opened terminal is the control terminal. If the terminal does not have a process group associated with it (as will be the case if this is the first instance of the terminal being opened), The process group id is made to be the process id of the opening process; this id gets associated with the terminal by storing it in the tty entry of the control terminal. If the terminal had a process group already associated with it, that process group id remains the same. The resulting process group id is also stored in the process table entry corresponding to the opening process, to be inherited by all of its descendents. In this way the opening process becomes the process group leader.

The above association will only take place if the opening process does not have a process group already associated with it (as in the case of getty). This prevents a regular user process (which resulted from the user shell) to become the process group leader when it opens another terminal line. If this were not the case, user processes that open different terminal lines would have their process group id changed, and the whole purpose of the process group concept would be defeated (see below). As will be brought out later, the getty process becomes the login shell and therefore the login shell is the process group leader. It controls every process initiated at the terminal. Since the process group id is inherited through the fork system call, every descendant of the shell has the same process group id.

In particular, when interrupt and quit characters are received from the terminal, and when the user hangs up, the corresponding signal is sent to every process with the same process group id as the one associated with the terminal *(gsignal)*. By default most of these processes exit as a result of the received signal. In this way user processes are not left around when a user suddenly hangs up the line. Nevertheless, some of these processes may have been set to ignore the hangup signal and will continue executing. To prevent these processes from receiving unwanted signals from the next terminal session, after a hangup the terminal is disassociated from the process group so that processes in that process group can no longer receive signals originating at the terminal. The new line discipline, which provides additional job control features, also distributes the stop signal to the process group when the stop character (usually ctrl-Z) is entered at the terminal.

3.6 Terminal Settings

As was mentioned in section 3.4, a terminal line behaves according to its settings. The ioctl system call is used to customize terminal lines to meet the appropriate requirements of the particular device attached to them, such as regular terminals, graphics displays, etc. The ioctl system call manipulates the relevant fields (depending on the command) in the tty entry corresponding to that terminal line.

Among other things, each tty entry contains four structures which contain settable values for each port. A description of these structures is contained in the discussion of the tty special file in reference [5]. The sgttyb structure contains flags for setting baud rate, terminal delays, terminal modes (RAW, CBREAK, etc), echoing, and parity. The tchars structure contains the special character settings such as interrupt, quit, etc., that are defined for both the old and new terminal interfaces. The local mode word contains flags for specifying such things as the erase mode, and 7- or 8-bit character input. And finally, ltchars sets special characters that are defined only for the new terminal driver. Of the four structures, the information contained in the sg_flags field of sgttyb and that in the local mode word is of special importance in establishing proper communication with nonstandard devices.

If *in* parameters are specified in the ioctl request (section 2.3.4), the kernel transfers the number of bytes specified in the request argument pointed to by argp from user to kernel space *(copyin)*. With this data it then updates the necessary fields in the tty entry. If request is one which requires modification to the actual terminal multiplexer's registers (i.e. the ones involving speed, parity, 8- or 7-bit character length, and modem control bits), those registers are updated also. If *out* parameters are specified, the kernel extracts the requested information from the tty table entry and copies it to the user area pointed to by argp *(copyout)*. The reader should refer to the ioctl and tty documentation for a complete description of the settable parameters that are available to customize a terminal line [5].

3.7 Reading from a Terminal

When a process reads from a terminal, the terminal multiplexer read/write routine is called (again using the cdevsw) after the uio structure has been properly initialized. The routine uses the minor device number as an index into the ttytable and invokes the line discipline specific read/write procedure (figure 3.1) with the tty table entry and uio structures as arguments. This routine takes appropriate action depending on the terminal mode.

In raw mode, the input raw queue is scanned for characters to be read and if present, characters are transferred one at a time to the user address space *(getc, ureadc, subyte)* until the characters in the queue are exhausted or the user requested amount is satisfied. Full 8-bit characters are passed. If no characters are present two actions may be taken: If the terminal was set in non-blocking mode, the read system call just returns the value zero, indicating that no characters were read and the global variable errno is set to EWOULDBLOCK indicating no input present; otherwise, the reading process is put to sleep on the event that a character is entered at the terminal (the address of that terminal's raw queue). The process will then be awakened by the interrupt service routine when a character is received.

The same thing happens in cbreak mode, except that 7-bit characters are given to the user, unless the PASS8 flag has been specified, in which case the full eight bits are passed. This is about the only similarity between cbreak and raw mode. As stated in the terminal modes section, the amount of input processing varies greatly in the two modes. However, as will be explained shortly, this processing takes place at the interrupt service level, right when the character is received and before it is placed in the input queue to be given to the user. The exception to this is the delay suspend character, which is processed only in the new line discipline read routine, and when in cbreak or crmod. This character causes the stop signal (SIGTSTP) to be sent to every process in the process group associated with the terminal, but in contrast to the suspend character, this action is delayed until a process attempts to read from the terminal.

In crmod things are done differently. First of all, the canonicalized queue is scanned for characters rather than the raw queue. As in the other two modes, if no characters are available the process will sleep waiting for a character. Again, if the terminal is in non-blocking mode, the process will not sleep but the read system call will return the value zero and errno will be set to EWOULDBLOCK. Secondly and most importantly, the sleeping process will not be awakened by the interrupt service routine as soon as any character is received, but only when a line terminating character is received. This is how lines of input are collected (in the raw queue) and edited before making them available to reading processes by transferring them to the canonicalized queue *(canq)*. When characters become available in the canonicalized queue, they are transferred one at a time to the user address space *(getc, ureadc, subyte)* until the line delimiter character is found or until the user specified amount is satisfied. If more characters were placed in the queue than the user requested, they will remain there for the next read operation.

As an aside, when processes are awakened, care should be taken that the conditions that put them to sleep no longer exist. This is to prevent race conditions to cause unexpected results. For example, suppose that two processes are reading from the same terminal but no input is available for either process. Furthermore, suppose the terminal is operating in raw mode. Since both processes are sleeping on the same event, both of them will be awakened when a character is received. The one that is scheduled first will get the character. However, the other one has no character to get and return to the user. Consequently, the appropriate thing for these processes to do upon awakening is to check again that a character is in fact present and if not, sleep again.

When reading from a terminal, the character count specified by the user in the read system call will not necessarily be satisfied. In most cases, the process will *sleep* awaiting terminal input. In raw or cbreak mode, as soon as one character is entered the process *wakes up* and the read system call returns with only that character being read. In crmod, the read system call returns as many characters as are found before a line delimiting character, or when the user specified amount is reached.

3.8 Writing to a Terminal

When a process writes to a terminal the line discipline write routine is invoked in the same way as the read routine. The user data, with base address and character count specified in the uio structure, are transferred from user to kernel space *(uiomove, copyin)*. Once the data are in kernel space, they are processed according to the terminal mode and flags before they are placed in the terminal's output queue *(b_to_q, putc)*, from which they will be transmitted to the actual device. First of all, the number of characters currently in the queue is checked. If it exceeds the high water mark, the writing process is put to sleep on the event that the output queue empties (the address of that output queue). The process will be awakened later when the character count has drained below the low water mark (the high- and low-water marks depend on the output speed of the terminal line).

If the terminal is in raw mode or if the LITOUT flag is set, all output translations are suppressed and the characters (8 bits) are placed directly in the output queue. Otherwise, 7-bit characters are used and output processing is done. If the cbreak bit is not set (the terminal is in crmod), EOT (normally ^D) characters sent to that terminal are stripped off (never put in the output queue) to prevent certain terminals from hanging up. If tab expansion is specified, tabs are expanded either in cbreak or crmod and the corresponding delay is generated. The driver also provides mapping of all characters to upper case and mapping of the character '~' to the character '`' for terminals that require so. The LCASE and TILDE flags should be set for these terminals respectively. Finally, if the crmod bit is on, newline characters are translated to <cr><lf>. Proper delays are generated also. Some additional output translations are provided when echoing received characters, such as translating the erase character into a <bs><sp><bs> sequence for crt erasing.

As will be seen later, these output translations can cause some unexpected results if the proper mode and flags are not used for the particular application. Once the characters have been processed and put on the queue, the transmitter is started to start transmitting the characters to the appropriate device. If there were any processes sleeping because the queue was full (above the high water mark), the transmitter *wakes* them *up* when the queue has drained below the low water mark.

3.9 Terminal Interrupts

When an interrupt occurs, the terminal multiplexer's interrupt service routine is invoked by picking its entry point from the particular vector address. Of course, every device (i.e. dmf) has its own set of vector addresses. The interrupt routine sets up a parameter to be passed to the general dmf interrupt service routine, to allow it to identify the particular dmf that caused the interrupt. The dmf itself identifies which of the eight lines caused the interrupt. In the case of an input interrupt, a dmf has a silo in which it can accumulate a small number of characters to be serviced during one interrupt. Embedded in these characters is the particular line that they came from, along with status information such as whether a parity or framing error occurred, or whether the silo overflowed. In this way, the dmf interrupt service routine grabs all the characters present in the silo and passes them to the specific line discipline interrupt service routine of each terminal line.

A transition in the carrier signal interrupts in the same way as an input character. If the line was not configured for full modem control (such as hardwired lines) this transition is simply ignored. Otherwise, action is taken depending on the state of the line. If carrier is now present and the line was waiting to complete the open (such as in getty), the state of the line is set to indicate so and sleeping processes are awakened; if the line was already open and it is doing flow control depending on the carrier state (the MDMBUF flag indicates this), the transmitter on the line is restarted. If the line loses carrier two things can happen: if the loss of carrier was due to flow control, the terminal multiplexer's stop procedure is invoked for that terminal line; otherwise, the loss of carrier is assumed to be the result of the user hanging up which causes the line to be turned off and the hangup signal to be sent to the process group associated with that terminal line. If no transition in carrier is detected (either because carrier was already present or because the line is hardwired), the received characters are checked for errors before they are passed to the line discipline receiver interrupt routine. If a parity error occurred and the user indeed specified only one of odd or even parity, the character is discarded. If a framing error occurred (which might have been as a result of a BREAK character) a null is generated if the terminal was in raw mode or an interrupt character (usually ctrl-C) is generated otherwise. As will be seen later, getty switches baud rates if it gets a null or an interrupt. If a silo overflow occurs a small warning message is logged on the console. The resulting character is passed to the line discipline receiver interrupt routine to be processed.

As discussed in the terminal modes section, the amount of input processing varies according to the mode the terminal line is operating on. In raw mode input characters are put directly in the raw queue. Processes awaiting input from this terminal are *awakened.* In cbreak or crmod, if the PASS8 flag is not set the 8th bit (parity) is stripped off. If the terminal was in the literal-next state (as a result of the previously received character being the literal-next character), the character is not interpreted but just put in the input queue (literal-next is only implemented in the new line discipline). If the stop character is received, the terminal multiplexer's stop procedure is invoked to disable further transmission interrupts on that line. When the start character is received the transmitter is reenabled. Actually, the transmitter will be restarted when any character is received in the line. To prevent this and force the line to wait for the start character before it can be restarted, the DECCTQ flag must be set. When interrupt and quit characters are received the corresponding signal is sent to the process group associated with that terminal line. The new line discipline provides additional control characters such as literal-next, flush-output, and suspend characters. The literal-next character works as explained above, the flush-output character flushes any pending output to the terminal and the suspend character distributes the stop signal to the terminal's process group. None of the control characters are put in the input queue, they are simply interpreted and the necessary actions are taken.

In cbreak mode characters are given to the reading process as soon as they are received. Thus, none of the line editing characters have any meaning in cbreak mode. Consequently they are simply put in the input raw queue as if they were regular characters. Processes that had been waiting for such an event are awakened. In crmod the line editing characters cause the appropriate action to be taken. The erase character removes the previously queued character from the raw queue. The kill character causes all the characters currently in the queue to be removed. If the CRTKILL flag is set, characters are erased from the screen with <bs><sp><bs> sequences. Otherwise a new line character is simply echoed to the terminal. The erase and kill characters can be escaped with the backslash character. In this case they have no effect and are simply put in the input queue. The new line discipline provides a word erase character and a reprint line character. The first one dequeues characters until a word delimiting character (blank or tab) is found. The latter reprints the current line of input, which is useful when such a line has been corrupted by a program outputting characters to the terminal. Normal characters are simply put in the raw queue until a line delimiting character is detected. When this happens, the updated raw queue is transferred to the canonicalized queue and processes *sleeping* on this event are *awakened.*

Characters are echoed back to the terminal unless the echo flag is turned off. Control characters are echoed differently depending on some control flags (such as CTLECH, CRTERA, etc.). There are some special characters like the start and stop characters, that are not echoed even if the echo flag is on. If they were, they could corrupt output sent, for example, to a graphics display.

3.10 Flow Control

There are two ways in which a terminal line can perform flow control. Lines with full modem control can do flow control on carrier signal, for which the MDMBUF flag should be set in the particular tty table entry. All lines can do flow control using the start (xon) and stop (xoff) characters. Most devices are manufactured so that they can either drop DTR (or one of the RS232 lines) or send an xoff character when their buffer fills up beyond a certain threshold. When the buffer empties below the lower threshold the device either activates DTR or sends the xon character. When a given line loses carrier (as a result of the attached device dropping DTR), or when it receives the xoff character, the kernel invokes the terminal multiplexer's stop procedure to disable further transmission interrupts on the given line. The process that was sending output to that line will eventually fill up that terminal line's output queue and will be put to *sleep.* When carrier is detected again, or when the xon character is received the kernel restarts the transmitter on the given line. The transmitter reenables transmission interrupts and once the output queue has drained below the low water mark, waiting processes are *awakened.* The cycle repeats until all data are transferred.

4. User Interface

The user interacts with the UNIX operating system through the user *shell.* This part describes the processes that set up the user environment and initialize the terminal line before the user starts working on the system. For full details about what these processes can do, the reader should refer to their specific manual pages [7].

4.1 Init

The *init* process is invoked as the last step in the boot procedure. When in multi-user operation, init creates a getty process for each terminal line in which a user may login, and goes into an infinite loop waiting for a death of child signal. The getty process opens and initializes the terminal line, reads a login name, and invokes login. The login process reads the user password, verifies it, and gives the user a shell. Eventually, the shell terminates as a result of an end-of-file or because of a received signal. Since the getty process had turned itself into the shell process, the shell is a child of init. When the shell exits it sends the death of child signal to its parent (init). Init wakes up, identifies the particular line and creates another getty to reopen and reinitialize the terminal line.

Getty does not create another process (does not fork) when it invokes login, but instead overlays itself with the image of the login program through an *execv* system call. The login program does the same thing; once it verifies the user password, it overlays itself with the image of the login shell.

Init reads the file */etc/ttys* and creates a getty for every line whose status is on. From that same line, it also gets the parameters to invoke the getty program with. Of particular importance is the first argument after the program name (getty). If present, getty will use this argument as an index into the *gettytab* database, from which the terminal specifications will be obtained. These terminal settings will overwrite previous settings obtained from the default entry. Further logins can be prevented on a particular line by changing its specific entry in the file */etc/ttys*, and sending a hangup signal to init. Init interprets this signal to mean that the file */etc/ttys* should be read again and terminates any processes associated with the terminal line whose entry has been turned off or no longer exists [8].

4.2 Getty

The *getty* process opens and initializes a terminal line. The specific device file to open is passed as an argument from the init program. Getty opens this file and executes two consecutive *dup* system calls. In this way, file descriptors 0, 1, and 2 share the same instance of the opened file, with a reference count of three. Since getty becomes the shell, the shell has these three descriptors already allocated. Furthermore, every user process is created directly or indirectly from the shell through the *fork* system call (recall that the fork system call increments the reference count on all shared descriptors). This implies that when the user closes descriptors 0, 1, or 2, (s)he would be merely decrementing the reference count on that instance of the file. This guarantees that the device closing procedures will be invoked only when the shell exits (closes all of its descriptors). In this way terminal settings will remain as long as the shell is executing (unless the user purposely decides to change them with ioctl calls).

The getty program reads the gettytab database to get the characteristics of each terminal line. It first sets global defaults defined in the default entry for all terminal lines. If a type argument was passed from the init program, getty also reads that entry in the gettytab database to overwrite default settings. The getty program issues different ioctl requests to set the default and specific characteristics obtained from gettytab. It sets the input and output speeds, parity, line discipline, the value of the different control characters, the terminal modes, and so on. For dial-up lines, the nx field in gettytab is used to change baud rates upon receipt of a null or interrupt character (this can happen as a result of the user hitting the break key). Parity can be set to accept either odd or even parity on input. If only one is specified, characters with the wrong parity are discarded. On the other hand, if both or none are specified, either parity is accepted. Even parity is generated on output, unless the odd parity bit is set and the even parity bit is cleared, in which case odd parity is generated.

The line discipline is set to the old line discipline. The login program will change this to the new line discipline if the user logging in has a C shell. The different control characters are set to their default values, unless otherwise specified in the gettytab entry. If for some reason these values are modified in the gettytab entry (and therefore remain for the life of getty), the login program will set them back to their default values to be used during the shell. There are three different sets of terminal modes that are used during the getty process. The first one is used to print out a banner and a login message. The second set is used while getty is reading the login name. Finally, the third set is used when getty has read the login name to leave the terminal state properly set for an interactive session (the shell). After getty has initialized the terminal line and gotten a login name, it invokes the login program.

4.3 Login

The *login* program, as mentioned above, resets all the control characters to their default values. Also, if the user login in has a C shell (as specified by the last field in the entry for that user in the password database), the line discipline is set to the new line discipline. The main role of the login program is to read the user password, verify it against the password database, and invoke the user shell. The login program sets the user and group ids of the process as taken from the password database entry for that user. It also initializes the basic environment variables such as the user's home directory, the type of shell, the type of terminal (as specified in the */etc/ttys* file), the user name and a default path. Once login has done its task it invokes the user shell.

4.4 The Shell

The login *shell* is the process through which the user interacts with the system. The shell could be seen as a command line interpreter. It goes into a loop in which it prompts the user for a command, reads the command, and executes it. The shell finishes the loop and exits when it receives the end of file character (usually ctrl-D). The shell can also terminate because of a received signal, such as the hangup, terminate, and kill signals. To accomplish the above task the shell reads a line of input in which the first token is the command to execute, and the rest are the arguments to invoke the command with (piped commands are exceptions to this rule). The shell executes a *fork* system call, and the child process overlays itself with the image of the specified command *(execv)*. The parent process (the shell itself) waits for a death of child signal before prompting the user for the next command. When the child process (the command) exits it sends the death of child signal that the shell had been waiting for, and the user is prompted for another command.

There is an exception to this sequence that occurs when the user specifies that a command should be run in the background. The user does this by putting an ampersand (&) at the end of the command line. The only difference in this case is that the shell (the parent) will not *wait* for the command (the child) to finish. Instead, it will prompt the user right away for another command. In this way a user may have several processes running at the same time. Another way this can happen is when processes created by the shell create *(fork)* child processes of their own.

The shell controls every process initiated at the terminal by means of the process group. Every process created at the terminal directly or indirectly by the shell will inherit the process group id of the shell. In this way every process initiated at the terminal will be in the same process group as the shell. Furthermore, this process group was associated with the terminal in the line discipline open procedure. When signals originate at a particular terminal, the kernel extracts the process group from its tty table entry and sends the signal to every process that has that same process group id *(gsignal)*. Of course, the shell itself must ignore interrupt, quit, and stop signals to prevent these signals from terminating it [8]. On the other hand, the hangup signal is considered to be the result of the user hanging up the line. Thus, the shell is terminated along with every other process in the process group.

5. Modifications

Terminal lines may be used for different types of devices, such as regular terminals, graphics displays, printers, plotters, etc. The amount of I/O processing needed for a given line depends on the device attached to it. For example, in the case of a regular user terminal, input characters like delete, kill, interrupt, and so on are available to make the user interface more friendly. Also, output characters like <cr> and <nl> are mapped to <cr><nl> to make output more readable. However, for other type of devices like plotters and graphics displays, this I/O processing would obviously cause disastrous results. For this reason, serial lines need to be customized according to the device that is attached to them. The three major terminal modes: crmod, cbreak, and raw, set the base level of I/O processing. Other flags in the tty table entry such as LITOUT, PASS8, CRTERA, etc., and a set of control characters prevent or provide additional I/O processing. This part describes the problems incurred by several devices connected to terminal lines, and modifications employed to solve these problems.

5.1 Existing Problems and Solutions

5.1.1 Flow Control

Several problems were encountered in generating flow control and primarily involved the use of *raw* mode for devices such as plotters and graphics displays that did not need any output processing. These devices should receive the user generated data just as they are, without being disturbed by the terminal driver output capabilities. Raw mode seemed to be the ideal mode for these terminal lines. However, since no characters are interpreted on input either, software handshaking is not possible because start and stop characters have no special meaning in this mode. Therefore, unless the device can process the data faster than it is being sent, or invokes flow control via hardware handshaking, raw mode is practically useless.

Crmod would solve the flow control problem, but it would introduce the problem of output processing being performed. As a result, EOT characters (normally ctrl-D) that could legally represent a graphics parameter, would get stripped off by the terminal driver and the device would never see them. Similarly, newline characters would be converted to a return-linefeed pair that would obviously not produce the intended result.

The solution is to use *cbreak* mode. Software flow control is invoked, and most of the above output processing is avoided. Many times the remaining output processing may still cause problems and must also be disabled. This can be accomplished by setting the appropriate parameters in the tty entry, and is described in the next section.

5.1.2 Tty Parameter Settings

The tty table entry contains flags which enable and disable certain terminal control processing settings. (See section 3.6.) Selected values are given in this section, however, for a complete description of each flag, refer to documentation on the tty special file [5].

In crmod and cbreak modes, eight-bit characters can be used on input by turning on the PASS8 bit in the local mode word. Similarly, eight bits could be used on output without any output translations (just like in raw mode) by setting the LITOUT bit. In addition, in cbreak mode, all control characters except the start and stop characters can be individually disabled to prevent distortion of specialized terminal commands.

Initially, it was not clear if these control characters should in fact be disabled or if they could be left set to their default values. The answer was that these control characters could be enabled for output only ports without causing any disturbance on that port's output. The reason is that all control characters are processed on input, not on output. Most output devices (i.e. plotters) cannot generate any of these characters in the first place. Even if a device used as an output device can input characters (such as a graphics display with a keyboard), they won't corrupt that port's output queue as long as they are not echoed. Echoing can be disabled by resetting the ECHO flag in sg_flags. The line editing characters (erase, kill, word-erase, etc.) do not have any meaning in cbreak since line editing is not possible. Remember that in cbreak mode every character is given to the reading process as soon as it becomes available. Furthermore, characters that generate signals (interrupt, quit, and stop) will not cause any problem because there won't be any process group associated with this port. This is because output ports are normally opened by user processes which are descendants of that user's shell. As explained previously, these processes will not become process group leaders because they already belong to the process group of the shell. Moreover, line printer ports opened by the line printer daemon *(lpd)* don't have this problem either. The reason in this case is because the line printer daemon disassociates itself from any terminal through an ioctl system call. Thus, it cannot receive signals originating at any terminal line.

5.2 Ports Controlled by Getty

5.2.1 Alphanumeric Terminals

The *getty* process employs three terminal modes while printing the login message, reading the login name, and setting the terminal state for the shell after login (see section 4.2). The operation of an alphanumeric terminal was entirely satisfactory after login, and after the corresponding mode settings (crmod, etc.) were invoked. Before login, however, a problem existed because the terminal line was given a default setting of raw mode by getty. Output redirected to these terminals would be corrupted because of the lack of flow control and ouput processing. No flow control meant that a subsequent loss of data occurred. Moreover, no output processing meant that lines of output were hard to distinguish from one another because they were not terminated by a <cr><nl> sequence.

Both problems were solved by using crmod instead of raw mode. However, since the getty program echoes the input characters itself, instead of letting the kernel do it, characters need to be available to the getty program as soon as they are typed. Therefore, cbreak mode was used also. If cbreak mode were not used, the getty program would not get any character of the user's login name until a line delimiting character was entered.

Consequently, the login name could not be echoed until it was completely typed, possibly misleading the user to believe that the terminal was not receiving input. These two bits, along with other appropriate flags such as new line and tab delays were set using the f1 field in the gettytab entry, which specifies the set of flags to be used while reading the login name (section 4.3). The gettytab entry is specified in */etc/ttys*, with one entry for each terminal.

5.2.2 Nonstandard Terminals

These types of terminals were mostly used as output devices. However, since the devices could also be used for input, it was decided that a user could login on those ports thus requiring a getty process. An example of such a device is an AED graphics display, which also can be used as an interactive terminal. For these terminal lines, a new *gettytab* entry was created, and the */etc/ttys* file was modified accordingly to contain the name of this entry. In this entry, a different set of flags was used while reading the login name (f1 field of the gettytab entry). Cbreak mode was used to prevent EOT characters from being stripped off the output sequence, and to achieve flow control. The crmod bit was turned off to prevent <cr> and <nl> to be mapped to <cr><nl>, as were the flags for tab expansion. These settings prevented graphics commands from being corrupted by unwanted output processing. All the above output processing could also be suppressed by simply enabling the LITOUT bit in the local mode word. As it turned out, this bit was necessary to be able to send 8-bit data to the graphics display anyway. This last setting required a minor modification to the getty program to actually update the flags in the tty entry with the local mode word just before reading the login name.

All the control characters except the start and stop characters were disabled since they would be of no use anyway when sending output to the display. This was done by setting the appropriate fields in the gettytab entry to the octal value 377 (negative one). As explained in section 4.3, when getty obtains a login name it resets the mode flags to be left for the login process. Also, the login process (section 4.4) resets the control characters to their default values to be used in an interactive session (the user shell). In this way, these terminals can be used as a graphics display while waiting for a user to login, and then have their settings changed to be used as a regular terminal once a user logs in.

5.3 Line Printer Ports

Line printers are handled by the line printer daemon. The line printer daemon opens and initializes the line printer ports. Since line printers are attached to terminal lines, the same underlying communication parameters apply. The line printer daemon reads the *printcap* database to get the characteristics of each line printer attached to a terminal line. The syntax of the printcap database is very similar to the syntax for the gettytab database. The reader should refer to the printcap documentation for further information about the printcap capabilities. The line printer interface is handled through the use of sockets, which are not discussed on this paper. Moreover, the line printer interface was not modified since it was already satisfactory.

5.4 Ports with Other Devices Attached

Along with ports with user terminals and printers handled by the *getty* and *lpd* processes, there were other ports that were not initialized by any process. These ports were connected to a raw device (no I/O processing needed) that did not require logging in, and therefore did not require a getty process. An example of such a device is a plotter. Originally, the user process would have to set the appropriate communication parameters before sending data to the port. However, this violated the UNIX philosophy that a user should be able to send information to any (suitable) file or device without any such intervention.

This problem was solved by spawning a process from */etc/rc.local* which initialized all the "raw" ports with their specific settings. (The rc.local script is invoked just before starting multi-user operation.) This process read a new database, */etc/rawtab*, to make the settings for every uninitialized port. The format of the rawtab database was made identical to that of gettytab. In our case, since these devices did not require as much I/O processing as regular terminals, the database was reduced to the

Consequently, the login name could not be echoed until it was completely typed, possibly misleading the user to believe that the terminal was not receiving input. These two bits, along with other appropriate flags such as new line and tab delays were set using the f1 field in the gettytab entry, which specifies the set of flags to be used while reading the login name (section 4.3). The gettytab entry is specified in */etc/ttys*, with one entry for each terminal.

5.2.2 Nonstandard Terminals

These types of terminals were mostly used as output devices. However, since the devices could also be used for input, it was decided that a user could login on those ports thus requiring a getty process. An example of such a device is an AED graphics display, which also can be used as an interactive terminal. For these terminal lines, a new *gettytab* entry was created, and the */etc/ttys* file was modified accordingly to contain the name of this entry. In this entry, a different set of flags was used while reading the login name (f1 field of the gettytab entry). Cbreak mode was used to prevent EOT characters from being stripped off the output sequence, and to achieve flow control. The crmod bit was turned off to prevent <cr> and <nl> to be mapped to <cr><nl>, as were the flags for tab expansion. These settings prevented graphics commands from being corrupted by unwanted output processing. All the above output processing could also be suppressed by simply enabling the LITOUT bit in the local mode word. As it turned out, this bit was necessary to be able to send 8-bit data to the graphics display anyway. This last setting required a minor modification to the getty program to actually update the flags in the tty entry with the local mode word just before reading the login name.

All the control characters except the start and stop characters were disabled since they would be of no use anyway when sending output to the display. This was done by setting the appropriate fields in the gettytab entry to the octal value 377 (negative one). As explained in section 4.3, when getty obtains a login name it resets the mode flags to be left for the login process. Also, the login process (section 4.4) resets the control characters to their default values to be used in an interactive session (the user shell). In this way, these terminals can be used as a graphics display while waiting for a user to login, and then have their settings changed to be used as a regular terminal once a user logs in.

5.3 Line Printer Ports

Line printers are handled by the line printer daemon. The line printer daemon opens and initializes the line printer ports. Since line printers are attached to terminal lines, the same underlying communication parameters apply. The line printer daemon reads the *printcap* database to get the characteristics of each line printer attached to a terminal line. The syntax of the printcap database is very similar to the syntax for the gettytab database. The reader should refer to the printcap documentation for further information about the printcap capabilities. The line printer interface is handled through the use of sockets, which are not discussed on this paper. Moreover, the line printer interface was not modified since it was already satisfactory.

5.4 Ports with Other Devices Attached

Along with ports with user terminals and printers handled by the *getty* and *lpd* processes, there were other ports that were not initialized by any process. These ports were connected to a raw device (no I/O processing needed) that did not require logging in, and therefore did not require a getty process. An example of such a device is a plotter. Originally, the user process would have to set the appropriate communication parameters before sending data to the port. However, this violated the UNIX philosophy that a user should be able to send information to any (suitable) file or device without any such intervention.

This problem was solved by spawning a process from */etc/rc.local* which initialized all the "raw" ports with their specific settings. (The rc.local script is invoked just before starting multi-user operation.) This process read a new database, */etc/rawtab*, to make the settings for every uninitialized port. The format of the rawtab database was made identical to that of gettytab. In our case, since these devices did not require as much I/O processing as regular terminals, the database was reduced to the

Crmod would solve the flow control problem, but it would introduce the problem of output processing being performed. As a result, EOT characters (normally ctrl-D) that could legally represent a graphics parameter, would get stripped off by the terminal driver and the device would never see them. Similarly, newline characters would be converted to a return-linefeed pair that would obviously not produce the intended result.

The solution is to use *cbreak* mode. Software flow control is invoked, and most of the above output processing is avoided. Many times the remaining output processing may still cause problems and must also be disabled. This can be accomplished by setting the appropriate parameters in the tty entry, and is described in the next section.

5.1.2 Tty Parameter Settings

The tty table entry contains flags which enable and disable certain terminal control processing settings. (See section 3.6.) Selected values are given in this section, however, for a complete description of each flag, refer to documentation on the tty special file [5].

In crmod and cbreak modes, eight-bit characters can be used on input by turning on the PASS8 bit in the local mode word. Similarly, eight bits could be used on output without any output translations (just like in raw mode) by setting the LITOUT bit. In addition, in cbreak mode, all control characters except the start and stop characters can be individually disabled to prevent distortion of specialized terminal commands.

Initially, it was not clear if these control characters should in fact be disabled or if they could be left set to their default values. The answer was that these control characters could be enabled for output only ports without causing any disturbance on that port's output. The reason is that all control characters are processed on input, not on output. Most output devices (i.e. plotters) cannot generate any of these characters in the first place. Even if a device used as an output device can input characters (such as a graphics display with a keyboard), they won't corrupt that port's output queue as long as they are not echoed. Echoing can be disabled by resetting the ECHO flag in sg_flags. The line editing characters (erase, kill, word-erase, etc.) do not have any meaning in cbreak since line editing is not possible. Remember that in cbreak mode every character is given to the reading process as soon as it becomes available. Furthermore, characters that generate signals (interrupt, quit, and stop) will not cause any problem because there won't be any process group associated with this port. This is because output ports are normally opened by user processes which are descendants of that user's shell. As explained previously, these processes will not become process group leaders because they already belong to the process group of the shell. Moreover, line printer ports opened by the line printer daemon *(lpd)* don't have this problem either. The reason in this case is because the line printer daemon disassociates itself from any terminal through an ioctl system call. Thus, it cannot receive signals originating at any terminal line.

5.2 Ports Controlled by Getty

5.2.1 Alphanumeric Terminals

The *getty* process employs three terminal modes while printing the login message, reading the login name, and setting the terminal state for the shell after login (see section 4.2). The operation of an alphanumeric terminal was entirely satisfactory after login, and after the corresponding mode settings (crmod, etc.) were invoked. Before login, however, a problem existed because the terminal line was given a default setting of raw mode by getty. Output redirected to these terminals would be corrupted because of the lack of flow control and ouput processing. No flow control meant that a subsequent loss of data occurred. Moreover, no output processing meant that lines of output were hard to distinguish from one another because they were not terminated by a <cr><nl> sequence.

Both problems were solved by using crmod instead of raw mode. However, since the getty program echoes the input characters itself, instead of letting the kernel do it, characters need to be available to the getty program as soon as they are typed. Therefore, cbreak mode was used also. If cbreak mode were not used, the getty program would not get any character of the user's login name until a line delimiting character was entered.

hardware communication parameters (speed, 8- or 7-bit characters, and parity) and the establishment of a flow control protocol. If hardware handshake is available (such as for lines with full modem control), the MDMBUF flag can be set to invoke flow control via the carrier flag. In this case, raw mode is employed with 8-bit characters and no parity. If 7-bit characters are desired, cbreak mode can be used instead. Of course, parity can be specified when using 7-bit data. For software handshaking, cbreak is used to allow the start and stop characters to be interpreted. If 8-bit data is required, the PASS8 flag is used for input, and the LITOUT flag for output. No parity is available in this case.

In our case, one port that was connected to an HP7475 plotter was modified in this manner. The hardware handshake method was tested to confirm the methodology, and required a cable with an extra wire to connect the DTR line (pin 20) of the plotter to the carrier detect line (pin 8) of the dmf port. The port was set to raw mode and communicated correctly and with proper flow control.

In spite of this success, software handshaking was preferred since a dmf multiplexer supports full modem control only on two of its eight ports. Thus hardware handshaking would require the plotter to be connected to one of those two ports. Consequently, software handshake was selected, and cbreak mode was used. The speed was set to 9600 baud. Both the PASS8 and LITOUT bits were turned on to allow for 8-bit characters required in graphics applications. A "de" field was added to rawtab to specify the device file name to be opened. Gettytab does not need this field since the device file name to be opened is passed from the init process to the getty process. Printcap also provides this field (although its name is different).

As stated in section 3.6 all of the control characters are automatically set to their default values when the first instance of the port is opened. Thus, it was not necessary to provide this capability in the rawtab database. The only characters of concern were the start and stop characters. As explained in section 5.1.2 the other control characters have no effect on the I/O processing of (primarily) ouput only devices. There was a question, however, as to whether enabled start and stop characters could conflict with hardware handshaking. Similarly, there was a question as to whether an enabled carrier flag could conflict with software handshaking. Neither case presented a problem for the following reasons:

1. Most devices are manufactured so that they can only do one form of handshake at a time. Therefore, as long as the device and the system agree on which protocol they are using, no conflict will occur.

2. Even if there exists a device that can do both protocols at the same time (say drop DTR and send an xoff character), no conflict will occur. This is because either the change in carrier state or the receipt of the xoff character will be serviced before the other. The one that is serviced first will set the terminal state to stopped and will disable the transmitter on that line. The other will find the state of the terminal line already stopped and will have no effect, just as when typing two consecutive stop characters at a terminal, the second one has no effect. Similarly, when carrier is detected and the xon character is received, the first one will clear the stopped state and will restart the transmitter. The other one will "clear" the already cleared stopped state, and will "start" the already started transmitter.

This approach of running a process from */etc/rc.local* may have a possible drawback compared to *getty* and *lpd*. The problem would occur if a user intentionally or accidentally changed the settings on a given line. In the case of getty, if the user changes any parameters from the shell, the line will return to its normal state when the shell exits and init creates another getty on that port. Line printer ports do not have this problem because the user does not have direct access to the parameters of these ports. If a user modifies any raw port parameters, however, they must be manually reset. Consequently, the user must leave the port in the same state as it was initially. This problem is not considered to be too serious since user intervention is no longer needed for proper operation of these lines, except for very special circumstances.

6.0 Conclusions

This paper presented the 4.3 BSD UNIX terminal line interface. Although many of the principles and algorithms are the same as for other UNIX versions, some variations exist. Of course no source code has been included due to copyright restrictions. The two major problems, flow control and lack or excess of I/O processing were solved for every terminal line. The solution consisted of specifying the right parameters for each line in the provided databases *gettytab* and *printcap*, and in the newly created database, *rawtab*. For the latter, a new process was created to actually read these parameters and initialize the specified ports. Although it was initially believed [9] that a kernel modification would be required to prevent special control characters from being automatically reset upon opening, this was not necessary since these characters have no effect on output ports.

The terminal interface has been handled in such a way that the user does not have to set the parameters for any port. All that is necessary is to open the port, if it has not already been opened by the system, and send data.

References

1. Bach, M., *The Design of the UNIX Operating System,* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1986.

2. Greer, K., "Writing UNIX Device Drivers," *Tutorial USENIX Technical Conference,* pp. 1-18, Dallas, TX, Jan 1985. *UNIX Programmer's Reference Manual,* Section 2, Computer Science Division, Department of Computer Science and Electrical Engineering, University of California, Berkeley, CA., 1986

3. Ritchie, D., "The UNIX I/O System," in *UNIX Programmer's Manual Supplementary Documents,* Vol 2, Section 5, Computer Science Division, Department of Computer Science and Electrical Engineering, University of California, Berkeley, CA., 1986

4. *UNIX Programmer's Reference Manual,* Section 4, Computer Science Division, Department of Computer Science and Electrical Engineering, University of California, Berkeley, CA., 1986

5. *UNIX System Manager's Manual,* Section 8, Computer Science Division, Department of Computer Science and Electrical Engineering, University of California, Berkeley, CA., 1986

6. Thompson, K., "UNIX Implementation," in *UNIX Programmer's Manual Supplementary Documents,* Vol 2, Section 4, Computer Science Division, Department of Computer Science and Electrical Engineering, University of California, Berkeley, CA., 1986

7. Kernighan, B., and Pike, R., "Signals and Interrupts" in *The UNIX Programming Environment,* pp. 225-232, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984.

8. Almada, A., "Enhancements to the 4.3 BSD UNIX Serial Line Interface," M.S. Thesis, The University of Texas at El Paso, El Paso, Tx, December, 1988.

Appendix: Internal Kernel Functions

nodev()
: Simply returns the ENODEV error code.

nulldev()
: Does nothing.

namei(pathname)
: Converts a pathname into a pointer to a locked inode. It uses several other algorithms beyond the scope of this paper.

openi(inode, mode)
: Invokes the specific open procedures for character and block special files.

sleep(event, prio)
: The process sleeps waiting for event. Wakeup will notify the process when event has occurred. When awakened, the process will enter the scheduling queue at priority prio.

wakeup(event)
: Scans the sleep queue and wakes up processes waiting on event

gsignal(pgrp, sig)
: Sends signal sig to every process with process group id equal to pgrp.

psignal(proc, sig)
: Sends signal sig to process proc. Called from gsignal.

subyte(addr, c)
: Transfers a character from kernel to user space. The value of c is placed into the user address addr.

bcopy(from, to, count)
: Copies the specified number of bytes within kernel space.

copyin(from, to, count)
: Copies the specified number of bytes from user to kernel space.

copyout(from, to, count)
: Copies the specified number of bytes from kernel to user space.

uiomove(addr, cnt, rw, uio)
: Transfers the specified number of bytes from user to kernel space (copyin), from kernel to user space (copyout) or within kernel space (bcopy), depending on the rw (read/write) and uio structure flags. One of the addresses to transfer the data from/to is addr and the other one is contained in the uio structure. The offset, count, and base fields of the uio structure are updated.

getc(clist)
: Gets the next character from the specified clist. Cbloks that become empty are returned to the freelist.

putc(c, clist)
: Puts a character at the end of the specified clist. If necessary, new cbloks are allocated from the freelist.

ureadc(c, uio)
: Transfers a character from kernel to user space (subyte) or within kernel space. Updates the base, count, and offset in the uio structure.

b_to_q(buf, count, clist)
: Copies the requested number of bytes from buf to clist in at most the size of a cblock chunks. This is because bcopy transfers bytes to contiguous memory locations. New cblocks are allocated from the freelist.

q_to_b(clist, buf, count)
: Copies the clist to buf in the same way as above. Empty cblocks are returned to the freelist.

canq(from, to)
: Transfers the characters in the from clist to the to clist. Uses algorithms q_to_b and b_to_q.

An Update on UNIX Standards Activities

Shane P. McCarron, NAPS International

This is the fourth in a series of articles on UNIX related standards activities. In this narrative I am going to cover a slightly wider area than usual. There have been developments at the ANSI X3 level, the National Institute of Standards and Technology (formerly the NBS), and within the POSIX committees which deserve attention. I will apologize at the outset for the length of this article, but I feel that all of the information is timely and important. In addition to information on group activities, included with each report is a contact person from whom you can get more information about these developments, and the names of USENIX Standards Watchdog Committee members through whom you can relay your opinions to the specific standards committees.

On the subject of the USENIX Standards Watchdog Committee, this series is now an activity of that group. Last quarter I used the article to solicit participation in the committee, and I am pleased to report that we have a number of new associate members. While I don't know everyone involved, I would like to thank those who have contributed: Anna Marie de Alvaré, Ted Baker, Mark Colburn, Doug Gwyn, Sol Kavy, Doris Lebovitz, Kevin Lewis, and Stephen Head. We are still in search of members for this group. While we will accept all comers, we are particularly interested in filling out our rather lean international input department. If you would like to be involved in the Watchdog activities, or know of someone who might be a good candidate, please contact:

John S. Quarterman
Texas Internet Consulting
701 Brazos, Suite 500
Austin, TX 78701-3243
(512) 320-9031
jsq@longway.tic.com

or

Mark Colburn
NAPS International
117 Mackubin St., Suite 1
St. Paul, MN 55102
(612) 224-9108
mark@naps.mn.org

C Language Standard

X3J11 (ANSI C standardization committee) met 26-30 September 1988 in Sunnyvale, CA. Principal business of the meeting was to respond to comments received during the third round of formal public review, which had closed earlier. In addition to the 15 letters formally registered with CBEMA's X3 Secretariat, 27 unregistered letters were included. There were 632 items contained in these 42 letters. In order to address them all, the committee was divided into response preparation subgroups, each of which tackled a subset of the total list of items. From time to time, the whole committee reassembled to hear issues that the subgroups were not able to completely resolve by themselves. In several cases a straw vote was taken to determine the sense of the committee. The resulting responses were formatted to produce the official X3J11 Response Document.

At the Sunnyvale meeting, several editorial changes to the draft standard were approved. The working definition of "editorial" was: A change is editorial if it clarifies the original intent of the committee; it is substantive if it changes the committee's intent.

There were several issues that were of particular interest to the UNIX/POSIX community:

• A change was made that clarified the ability of an application to portably reestablish a signal handler for the signal that caused entry to the handler. This is indeed allowed under the standard. The important passage reads:

If the signal occurs other than as a result of calling the *abort* or *raise* function, the behavior is undefined if the signal handler calls any function in the standard library other than the *signal* function itself (with a first argument of the signal number corresponding to the signal that caused the invocation of the handler) or refers to any object with static storage duration other than by assigning a value to a static storage duration variable of type *volatile sig_atomic_t*.

• IEEE Std 1003.1-1988 (POSIX) requires that the *fflush* function specified by X3J11 have some additional semantics. The committee confirmed that this was indeed allowed by ANSI C.

• The IEEE P1003.1 working group had asked X3J11 to consider making the symbol "environ" a reserved external identifier. This would mean that an ANSI C conforming portable application could not use the symbol. This request was made because in traditional UNIX implementations application launch routines initialize this variable to be a pointer to the user's environment variable list, and this may not be what a strictly conforming ANSI C application would expect. This issue was raised before the committee, but found no support for a change; the committee response for this was as follows:

The ANSI C and IEEE 1003.1-1988 standards are not necessarily in conflict here, although it is true that in order to avoid the name-space conflict a mutually conforming implementation must rely on some mechanism such as 'global symbolic equate' or a zero-size global object 'environ' in a separate library module immediately preceding the module that defines storage for '__environ' (the name used by the C run-time startup code). Implementor control over the way the linker operates, while inappropriate to require for the more universal C Standard (hence the constraint on uniqueness of external identifiers), is not unrealistic to expect for most POSIX implementations. Several implementors have in fact indicated their intention to provide such a feature.

Another solution, of course, would be to use separate run-time startup modules for strict ANSI-conforming and vendor-extended (possibly POSIX-conforming) implementations, perhaps via a compiler flag. This may be useful anyway, for hiding extensions in certain standard headers.

Because no substantive changes to the proposed standard resulted from the third-round review process, X3J11 voted unanimously to submit the standard as edited to reflect approved editorial changes to CBEMA X3 as the proposed ANSI C standard, pending completion of additional review as described below.

The draft Response Document was reviewed first by a small group of X3J11 members using electronic mail, then by a group meeting at Plum-Hall in Cardiff, NJ, on 20-21 October 1988. The responses were checked for completeness, consistency, and accuracy, and occasionally the original responses were changed to achieve those goals, or to meet the additional requirement that no unauthorized substantive change to the proposed standard could be promised by any response. Changes made at the review meeting were subsequently edited into the master Response Document. Two significant areas of the standard were affected by editorial changes resulting from the response review process: the description of pointer arithmetic was substantially reworked to avoid reliance on an assumption of byte addressability, and the specification of the role of type qualifiers was rewritten to clarify the significance of what was called the "top type" (now called "type category").

On 1 November 1988, the draft proposed Standard itself was reviewed by several X3J11 members in a meeting at Summit, NJ. Since the draft already contained the results of the Sunnyvale meeting and response review meeting, very few changes were found necessary at the meeting.

On 9 November 1988, the Rationale Document (designed to accompany the Standard) was reviewed by a group of X3J11 members meeting in Cambridge, MA.

On 14 November 1988, copies of all three documents (Response, Standard, Rationale) were express-mailed to the 15 X3-registered commenters, who had 15 working days (from November 18) in which to reply to X3 if they felt that their items were not properly addressed by X3J11. The commenters were encouraged to first discuss problems with X3J11 members, in hopes of reducing the amount of negative feedback to X3.

On 9 December 1988, all three documents plus any feedback from the commenters were to be submitted to CBEMA's X3 Secretariat as the official X3J11 proposal for the ANSI Standard for Programming Language C. After review by X3, assuming no problems arise, the proposed Standard will then be submitted to ANSI for official ratification as an ANSI standard. It seems probable that the final ANSI C standard will be published some time during 1989.

The USENIX Standards Watchdog Committee contact person in X3J11 is Doug Gwyn. He can be reached at:

Doug Gwyn
US Army Ballistic Research Lab
801 L Cashew Ct.
Belair, MD 21014
gwyn@brl.mil
+1 (301) 287-6647

National Institute of Standards and Technology

On August 30, 1988 (four days after publication of the previous in this series) the NIST published their Federal Information Processing Standard for POSIX. Suffice it to say that this FIPS is finally approved, but differs substantially from the approved IEEE standard in a few key areas. The NIST is now working to revise the FIPS so that it is more in line with the real standard. This new FIPS should be announced in the Federal Register in early January, and after time for public comment and review, will be formally approved. The NIST expects approval sometime in summer 1989.

In the last article I mentioned that the NIST had announced their intent to create FIPS in other areas. They have now released a preliminary FIPS for System Administration and are about to release one for Shell and Tools. They have also stated that by year's end they will release a FIPS on utilities with User Interfaces (like vi). While in the case of Shell and Tools the NIST is going to use Draft 8 of the 1003.2 standard, there are no existing formal standards in the other areas. Instead of waiting for standards bodies to develop mature documents, the NIST is going to a number of different versions of UNIX, and picking those things that look neat. The System Administration FIPS in particular is disturbing. There are a number of utilities in there from AIX (IBM's version of UNIX), Xenix (SCO or Microsoft, I can't tell), and of course the SVID (from AT&T). This ensures that there is no existing system that will conform to the FIPS on day one, and also shackles the newly formed IEEE working group on System Administration.

I really don't know what the NIST is trying to achieve. It appears they are working toward their stated goal of creating a full suite of specifications to flesh out the Applications Portability Profile (a conceptual model of portability specifications created by the NBS over the last few years). I used to think that they had some sort of hidden agenda, but I don't believe that any more. I used to think that they were trying to railroad standards to make sure that the government's needs were satisfied. In this I have also been proven wrong. They have now shown their ability to create standards at will, thereby invalidating the work of the standards bodies before they can even begin. This interesting turn of events proves that in their previous heinous acts they were just being nice. They could have superceded the process altogether if they had really wanted to!

It was bad enough when the work of the committees was being affected by the arbitrary timelines imposed by the NIST, but now they have created a framework within which any standard on, say, System Administration, will have to fall if it is to be taken seriously by the vendor community. What vendor in its right mind would conform to a formal standard that was not in line with the standard being required by all U.S. federal agencies? The obvious answer is "vendors that don't want to sell to the government." In other words – none. Moreover, what vendor sponsored committee member is going to propose something for a standard that would make their employer not be able to sell to the federal government? Again, none.

I have given the NIST an opportunity to rebut the comments made above, and they are in the process of doing so. I will publish their comments as soon as I have them available. However, I would guess that they will say something like "These are just first cuts. In the future we will modify the FIPS to conform to standards produced by standards making bodies." That's great, but it really doesn't help. First, it would be a disservice to the federal user community to force them to change from an environment in which they have become comfortable. Second, it is a mistake to assume that the vendors are going to want to conform to one standard for a while, and then change over later. If there is a standard that is being required by a substantial part of the user community, then that is the one to which vendors are going to conform. And if vendors conform to it, it then becomes the existing practice that must be formalized by standards bodies like IEEE P1003. It's a vicious circle, and in the end the losers are the users. They are being handed an ill-considered standard; one that is being foisted upon them just because some small group of people, after consulting with a handful of their (rather unique) user community, have decided that this is the way it is going to be.

In defense of the NIST, I know that they are not trying to destroy the standards making process. They are just a bunch of people trying to do their jobs the best way they know how. It is unfortunate that in doing so they may end up doing more harm than good.

The USENIX Standards Watchdog Committee has no contact person with the NIST. For further information on NIST activities you can contact me or Roger Martin.

Roger Martin
National Institute of
Standards and Technology
Software Engineering Group
Room B266
Technology Blvd.
Gaithersburg, MD 20899
rmartin@swe.icst.nbs.gov
+1 (301) 975-3295

1003.0 – POSIX Guide

At this meeting of 1003.0 the group was presented with the first working draft of the guide document. Throughout the week the committee met in both small groups and in plenary sessions to expand on the first draft and start nailing down the exact focus of the project. In particular the group concentrated on the issues that had been raised and entered in the Issues Log, the overall objectives and the scope of the document. The purpose of the discussions was in part to clarify the strategic goals of the committee, and in part to prioritize those items that have already been decided upon.

Each small group that met worked on a particular area of the draft, expanding on its contents. As the full working group could not decide on the level of detail that should be included in each section, it was left up to each small group and revisited later. Topics that are being covered include: the Benefits of Open Systems, Key Open Systems Areas.

The USENIX Standards Watchdog Committee contact for 1003.0 is Kevin Lewis. He can be reached at:

Kevin Lewis
DEC
Suite 645
1331 Pennsylvania Avenue NW
Washington, DC 20004
klewis@gucci.dec.com
+1 (202) 383-5633

1003.1 – System Services Interface

The big news from this meeting of the 1003.1 working group is that its Chair, Jim Isaak, has resigned after 5 years of work. Jim is also Chair of 1003, the convenor of the ISO work item on POSIX, and a passel of other things; consequently he felt that he could no longer contribute the amount of time to 1003.1 that is really necessary for a working group chair. I would like to take this opportunity to thank Jim for all of the effort he put in to making the first POSIX standard a reality. We are fortunate that there are people like him in the industry.

The new chair of the committee is Donn Terry. Donn has been co-chair for a couple of years now, and has been the real chair (if not in name, then in actions) since the standard went to ballot in November of 1987. He is one of the original members of 1003.1, and is also the chair of the US Technical Advisory Group on POSIX to ANSI. Donn coordinated the last two rounds of balloting on the 1003.1 standard, and did an excellent job. I'm confident that he will prove to be as able a chair as Jim.

Almost as important is that the standard is now available in print. The bound version of the standard, while almost unreadable because of IEEE enforced formatting changes, and hard on the eyes because of its ugly split-pea-green cover, is now available for $16 (members) or $32 (non-members) from the IEEE office in New Jersey. For a copy, please contact:

IEEE Service Center
445 Hoes Lane
Piscataway, NJ 08854
+1-201-981-0060

After electing the new chair, the working group got down to business. They continued their work on extending the first POSIX standard, IEEE Std 1003.1-1988. Their primary areas of focus are now a new archive format, a functional interface for terminal interaction, and cleanup of the first standard. In addition the group starting forming a sub group to be the interpretations committee for the released standard. Each standard must have a "supreme court" of sorts. Users of the standard may submit formal questions to the IEEE, and those questions will in turn be conveyed to the interpretations committee. It is up to this committee to figure out the answers to the questions, and then to modify the standard if necessary so that in future printings the question doesn't come up. More about this as it develops.

One issue of great import is internationalization of the standard. The international community has some concerns, particularly in the areas of character sets and the use of the words "byte" and "character." These concerns were in particular voiced by the Japanese representatives at the October meeting of WG15 in Tokyo. The committee tried to be very careful when drafting the standard, but apparently not everything was covered. In any event, the working group now has to write an appendix to the standard which specifies the intent of the group regarding international implementations of POSIX. The standard is not really an implementors guide, but the appendix should provide a better guide to the intent of the group. Hopefully this appendix will be enough to keep the international community at bay long enough for the standard to be ratified as an ISO Draft International Standard (DIS).

On a related note, the ISO Working Group for POSIX (ISO/IEC JTC1/Sc22/WG15) has recommended that DP 9945 (the draft proposed international standard POSIX) be elevated to a DIS. This means that the standard has to go through another (international) balloting period before it can be a real international standard. Personally, I don't anticipate any trouble.

The 1003.1 committee hopes to ballot a revised version of the standard within two years. This revised version would contain a new archive format, some additional functions there were left out of the original but are now felt to be necessary, and any clarifications that have come from the interpretations committee. In addition all of the interfaces in the standard will be described in a way that is programming language independent, and there will be a chapter that has the C language binding to this language independent description. It sounds like a big job, but the committee is optimistic. It is also small enough now that it might just get it done in that time frame.

I am the USENIX Standards Watchdog Committee contact for 1003.1.

1003.2 – Shell and Tools Interface

This working group never ceases to impress me. In September the group was given about three weeks to go over draft 7 of the standard and review it as if it were a formal ballot. This means that problems discovered in the draft must be reported to the committee using the formal POSIX balloting format, within the specified time limits, in order to be considered. A surprising number of people were able to work very hard and come up with about 1500 objections to the 600 page document.

Okay, so a lot of people, given 3 weeks, can really find a lot of problems with a somewhat immature document. Maybe not terribly impressive. Then a group of 40 people meet in Hawaii, not a place known to be conducive to work, and manage to review every single objection and resolve them! This is truly amazing, and I think everyone at that meeting (including myself) deserves a medal. Moreover, I would like to take this opportunity to publicly eat the words I wrote last quarter. They may just pull it off! The draft that goes out for balloting in the formal IEEE process is certainly in much better shape than the 1003.1 document was when it first went out. Also, P1003 learned a lot from the .1 ballot, and that knowledge should help make the balloting of .2 smoother.

Some specific changes of interest were:

• Based on a decision from the previous meeting and several balloting objections, the *fgrep* and *egrep* commands have been removed from the standard, and the functionality that they provide is being encompassed in the definition of *grep*. This new *grep* will have options *-E* and *-F* which will give it the exact functionality of *egrep* or *fgrep*, respectively.

• The draft has a command in it called *colldef*. *colldef* allows the portable definition of collating sequences, which can then be used by utilities that do string comparisons with the C Standard function *strcoll*. The theory goes that an implementation will provide applications with a means for creating collation sequence definitions (*colldef*), and then also allow the application to specify which collation sequence to use when calling utilities like *sort* (through the environment variable LC_COLLATE).

It all sounds pretty good, but the definition of *colldef* was so incomplete and confusing that some balloters suggested it be removed from the standard altogether. The definition of this utility now provides for a lot of additional functionality, and is much clearer than it used to be. While this part of the standard is not talked about much, I believe that it is probably the most important part. The international aspects of POSIX are sort of obscure, but they will allow for more portable applications, and also allow for some previously unheard of uses for utilities like *sort*.

• A closely related utility, *xform*, was placed in the standard to allow for the transformation of strings by a shell script just as can be done using the *strxfrm* function in Standard C. After much discussion in the small group, this command was removed from the draft. While there was some dissenting opinion, the majority thought that this would have very limited usefulness to a portable shell application. As I was the dissenter, I can say that I wanted it in because there is no other way to portably compare strings in the shell from an international perspective. If a user enters something and then later you want them to enter it again, you cannot portably compare those strings without the xform utility. Alas, you win some...

• An interesting development was the decision that the C language functions in the standard be moved into a chapter for C Language interfaces, and that their original position in the document be reserved for the language independent descriptions of some of the functions. In the end it may be that some of the functions are really not ones that need to exist in other languages, and as such should not be in the language independent section. This event is interesting because it shows the intent of this working group, and indeed all of the POSIX working groups, to describe their standards in a language independent manner. This was a requirement of the international community, and I am glad to see that it is being carried out.

• In what I consider a victory for the users of the world, the UUCP style commands in the standard have been moved out of the document and into an appendix. These commands, *uuxqt* and *uuname*, have been in the standard for about a year, but no one could really figure out why. As described there was no underlying transport mechanism or protocol defined, so they could not possibly have been reliable in any event. They were placed in the standard as a spear; something that you could throw out and have no idea if it worked or not. Depending on the feeling of the balloting group, these commands will either be fleshed out into a full definition of the UUCP "networking" system, or removed from the standard altogether.

• While the UUCP commands are gone, the message sending command *sendto* is still in the standard. This command allows an application to send text to an address with an implementation defined format to be deposited in an implementation defined location and delivered in an implementation defined manner. No kidding. That's what it says. It also used to say *sendto -r* would try to read from your personal implementation defined storage location, but that it might not do anything. Fortunately, the working group couldn't figure out a single reason why a portable application would want to read mail. While this is usually not enough cause to remove something from a standard, when coupled with the danger that it might not do anything if executed, the evidence seemed to lean toward removal. This option has been axed.

• There is now a section of the standard on application installation. Actually, there has always been a section for that, but until now it has been full of stuff that wasn't really worth reading. The new definition is a little bit complex, but it seems to be fine. It allows for an application, on installation, to determine what system resources are available, and to then sort of dynamically inform itself about them. There is also a system resource data-base, and all sorts of other neat stuff. I don't have a handle on all of it yet, so stay tuned.

There were all sorts of other changes made to the draft, but they are primarily editorial, and are of course all subject to review by the balloting group.

The schedule for balloting goes something like this: Assuming the document gets to the balloting group in mid-January, the period will close in mid-February. Then all of the received objections will have to be resolved or commented on, and it will be recirculated. This may happen several times before the document is finalized. Since each recirculation & resolution period takes 3 to 4 months, it could be early 1990 before we see a ratified standard.

In the meantime, since the working group doesn't have anything to do with a standard while it is going through balloting, work will progress on the new User Portability Extensions supplement. The idea here is that a supplement to 1003.2 will be released soon after the initial standard. This supplement will describe the traditional UNIX utilities that have user interfaces (e.g. *vi*). Note that the utilities to be described are the traditional ones, and have nothing to do with windowing-mouse interfaces. Work on that topic is progressing in other areas.

I am the USENIX Standards Watchdog Committee contact for 1003.2.

1003.3 – Testing and Verification

This POSIX working group met along with the others in Honolulu in October. The agenda included a status report on NIST activities, review of previously assigned action items, developing a strategy for future work with other P1003 (POSIX) working groups, revision of Draft 7.1 document, and assigning new action items.

Roger Martin (NIST & P1003.3 Chair) gave a status report on the current NIST FIPS and their Conformance Testing Policies for the POSIX FIPS. He stated that this "Initial" POSIX FIPS has been approved and they intend to revise the FIPS now that the P1003.1 Standard is finalized. The NIST Test Suite, PCTS, has been provided to NTIS (National Technical Information Service) for public distribution at a price of $2500 and is being distributed since September 5, 1988. Its distribution was awaiting FIPS approval. Roger Martin also presented a proposed schedule for a series of Application Portability Workshops sponsored by NIST. He described a workshop that had taken place in September 1988 covering Shell & Tools, System Administration, and X Windows. One of the areas to be covered in a future Application Portability Profile FIPS and workshop include the Terminal Interface Extension. The workshops are intended for implementors and users.

The remainder of the meeting concentrated on rewriting and restructuring the Draft 7.1 document, including test assertions.

During the week of meetings one small group of Test Assertion Reviewers continued to update the 1003.3 Draft 7.1 assertions.

Two other small groups concentrated on rewriting and restructuring 1003.3 Draft 7.1 document. One group's emphasis was the development of 1003.3 Generic Test Method chapters (i.e. terminology, testing levels, generic PCTS output). The second group's emphasis was in developing 1003.1 specific Test Method sections.

The P1003.3 group is gearing up for balloting this standard in early 1989. Each P1003.3 member is part of the "mock" ballot group, identifying and formulating any possible objections.

Future work of the P1003.3 committee was also addressed. The P1003.3 Working

Group wants to influence the other P1003 Working Groups into writing testable standards. To achieve this, a liaison program will be implemented to have members from P1003.3 working in a liaison fashion in each of the other working groups.

The P1003.3 working group Project Authorization (PAR) will need to be revised in order for the group to develop an overall Test Method standard and the development of specific standards for each appropriate 1003 activity.

The USENIX Standards Watchdog Committee contact for 1003.3 is Doris Lebovits. She can be reached at:

Doris Lebovits
AT&T
Rm 5-211
190 River Rd,
Summit, NJ 07901
lebovits@attunix.att.com
attunix!lebovits
+1 (201) 522-6586

1003.4 - Real Time Extensions to POSIX

In the past I have written some things about this committee that were pretty critical. I saw them as progressing too slowly to have the impact I hoped they would have. I know that nothing I wrote or said motivated them, but I am now happy to report the following: 1003.4 is almost ready to go to mock ballot! Apparently it all came together in the last couple of months, and they are now ready to ask a wider group for an opinion. They plan, at the January meeting, to go through all of their working papers and appendices, integrate them into the draft, and them submit it for a mock ballot before the April meeting. The results of the trial ballot will tell them how much more work they need to do before going to formal ballot. If all goes well, they should be able to ballot after the July, 1989 meeting. Given the way ballots tend to go, that would mean a completed standard in early to mid 1990. This is particularly exciting since dates in 1991 had been bandied about previously. Getting this standard out a full year earlier is astounding.

Many people are probably curious as to what is contained in a Real Time standard. Well, many things that didn't make it into 1003.1, for starters. Here is a partial list: Asynchronous I/O, Shared Memory, IPC, Asynchronous Event Notification, Process Memory Locking, Timers, Priority Scheduling, Semaphores, Synchronous I/O, and Realtime Files.

Some of these are going to be particularly contentious. In particular Events and Memory Locking could be a problem. The mock balloting should flush out these issues so it can be cleaned up before formal balloting in the fall.

The USENIX Standards Watchdog Committee contact for 1003.4 is Sol Kavy. He can be reached at:

Sol Kavy
Hewlett-Packard
19477 Pruneridge
Cupertino, CA 95014
sol@hpda.hp.com or hpda!sol
+1 (408) 477-6395

1003.5 - Ada Language Binding

This group is interesting. They have now distributed draft 1 of their standard to the working group, but they are very close to finishing.

The primary goal of the P1003.5 working group is to produce an Ada language binding for the operating system services interface defined by the P1003.1 standard. This work has progressed to the stage of circulating draft chapters within the group. These chapters are to be reviewed at the next .5 meeting (in January).

The last .5 meeting was 7-9 September 1988 in Minneapolis, MN. One of the issues discussed there was improving coordination with the rest of P1003. The last two P1003 meetings conflicted with major Ada meetings, so that .5 chose to meet separately. This has not been good for communication. Fortunately, there are no major conflicts with the Ft. Lauderdale meeting, and they will attempt to synchronize future meetings with the rest of the P1003 working groups.

Major issues which were discussed at the September meeting included: (1) the relationship of Ada I/O and POSIX I/O, and how this relates to P1003.0; (2) (missing) support for

Ada in the P1003.2 standard; (3) real-time features required by Ada, and whether P1003.4 will provide these; (4) changes to .1 between draft 12 and the final version that will require changes to the .5 draft chapters; (5) the relationship of Ada tasks to POSIX processes; (6) whether the organization of the P1003.5 document should mirror the .1 document.

One of the central problems they face is reconciling the relationship between Ada tasks and POSIX processes. Unlike POSIX processes, Ada tasks share a common logical address space. If they map Ada tasks onto distinct POSIX processes, they need a way to share memory and file handles (opened after fork) between processes, which is not provided in .1. (Support for shared memory is on the .4 agenda, but the final form remains uncertain.) Moreover, there are applications of Ada tasks that require task switching, creation, and termination to be performed much faster than may be possible for POSIX processes.

On the other hand, they might implement tasks as multiple threads of control within a process, but then they run into other problems. Unfortunately, multiple threads of control within a process cannot be supported well without some cooperation from the OS. For example, a blocking system call by one thread should not block other threads. For another example, what happens when one task is in the middle of a system call and another one forks? (For now, P1003.5 agreed that Fork/Exec should be allowed, but that their effects in a multitasking Ada program are implementation dependent.)

The concept of POSIX support for "lightweight processes" is appealing. The group will explore the applicability of such a solution. In order to broaden the base of interest, they have agreed to sponsor a "Birds of a Feather" discussion on this issue at the Ft. Lauderdale meeting.

Another major problem is reconciling POSIX signals with Ada semantics. The .5 group has done some preliminary work on this. The concept most closely corresponds to an asynchronous Ada exception, but this construct is of questionable legality. The legal Ada mechanism appears to be entry calls, but this presents other problems. Much work remains.

A third problem area is data representation, and character sets in particular. POSIX already has problems with international character sets, arising from special uses of certain glyphs, and from an implicit assumption that characters are represented as bytes. Ada makes this worse, since it specifies a very specific standard character set (ASCII). The .5 group proposes to recognize POSIX characters (and strings) as distinct from the Ada versions, and to provide transfer functions for situations where one must be converted to the other.

Due to a conflict with the ACM Tri-Ada conference, 1003.5 was not able to meet with the rest of the POSIX committees in Hawaii. However, several individual members volunteered to attend as liaison with the other groups. This will probably turn out to have been very helpful in resolving some questions about division of responsibility.

It became clear during the 1003.1 meeting that .5 should not feel constrained to mimic the C-language binding, but should move ahead boldly to create a true Ada interface. Further, it appeared that due to Ada's strong typing requirements .5 might be closer to a language-independent version of .1 (required by ISO) than the present .1 standard, and might well influence the form of the future .1.

Meetings with the .4 revealed that support for Ada's real-time requirements might be provided by that group, but not necessarily in a suitable form or soon enough. In particular, the subject of lightweight processes, which might be used to implement Ada tasks, is not on the .4 agenda. This leaves the subject open to be addressed by .5.

These, and observations by other .5 members serving as liaisons are likely to influence the direction of work when the group next gets together.

The USENIX Standards Watchdog Committee contact for 1003.5 is Ted Baker. He can be reached at:

Ted Baker
Department of Computer Science
Florida State University
Tallahassee, FL 32306

tbaker@ajpo.sei.cmu.edu
baker@nu.cs.fsu.edu
+1 904 644-5452

1003.6 – Security Extensions to POSIX

The 1003.6 committee met with the other POSIX committees in Hawaii. At this meeting they decided to divide the work into different groups. The groups were addressing: Audit, Definitions, P1003.6 Scope, DAC, and Privileges.

Each small working group met every day, and on the morning of the final day of the meeting a wrap-up session was held to update all the members of each working group's progress. The following information was presented:

Audit

1. Goals:

- Satisfy TCSEC Requirement.

- Reduce the amount of change to POSIX as much as possible.

- Primarily to make audit trail entries.

- Portability for audit administration & analysis packages and private applications.

- Audit Data Interchange Format.

2. Areas of Investigation:

- Definitions

- Event/Classes (what are they?)

- Pre/Post Selection Criteria

- SSO Interface

- Subsystem Interface

- Record/File Format

- IDs (audit ids,...)

3. Future:

- Detailed Input Requested

- Interim Event/Classes

- BNF for Audit Token Grammar

Note that the administration interface issues have been considered to be a HANDS-OFF right now.

Definitions

The following information was presented:

1. The structure of the definitions will be similar to 1003.1 structure: terminology section, conformance section, general terms, general concepts, and acronyms.

2. The draft 0 definitions were based on four documents: ISO, ECMA, IEEE Std 1003.1-1988, and the Orange Book.

3. The goal of this group is to assure that 1003.6 definitions are consistent and relevant to 1003.6 areas without overstepping or duplicating existing definitions from other 1003.x groups. In case some of the 1003.6 definitions conflict with 1003.X ones, the action will be to propose a redefinition of the term.

*P1003.6*Scope

The proposed Scope was discussed and the conclusion was that it needed reworking. The area of I&A was considered not addressed, as were trusted recovery (which the real-time people may need) and others. In the draft a lot of the issues that will not be supported right now are marked so because of lack of experience or not enough technical maturity. The important point is not whether we have the experience or not, it is to be aware of areas where users want security, areas where the committee thinks security should be provided, and point them out in the Scope. If areas become a problem later, they can be dealt with at that time.

For the next draft of the 1003.6 document, the table of contents will contain: Scope, Definitions, Feature Overview, Existing 1003.1 Functions, Existing 1003.2 Commands, Section for Each Feature, and an Appendix.

The Feature Overview covers a discussion, functional interface summary and command summary of each feature. Then in the feature section there will be the functions, commands, descriptions, and security specifications.

In the appendix there will be a rationale that maps to the document sections.

It was remarked that all the future features such as Networking and System Administration should be annotated in an appendix as areas that will be covered as extensions.

Discretionary Access Controls

This group was the one with the most activity, generating a lot of conflicting ideas even within itself. However, they did resolve to put together first the Rationale section of the document and work on the agreeable parts, then later debate the contentious ones. One of the conflicting topics was default Access Control Lists. This is probably needed, but apparently will not be within the scope of the standard.

Privileges

Privileges is a topic wrought with philosophy, and computer professionals love to be philosophers. In spite of this, definitions of privilege and certain types of privileges were completed. A paper from IBM was taken as a framework for the privilege section. During the meeting a few operations were identified as necessary, although the list is far from complete: *getpriv, setpriv, enable/disable_priv, droppriv*.

Another issue brought to the whole group was Internationalization, and the decision was not to address it as long as they can. This is unfortunate, as the charter of POSIX is to be as international as possible. The 1003.1 committee learned the hard way that internationalization cannot just be stapled on later. It must be in there from day one or it becomes extremely difficult to make it work. In the case of security, labeling is an area in which internationalization is a must. If it is not placed in there initially, it may never get in.

The upshot of all this is that the small groups produced the guidelines for the next meeting and the topics that are going to be covered in the near future.

This group has targeted mid-1990 for a complete draft ready to ballot. The USENIX Standards Watchdog Committee contact for this group is Anna Maria de Alvaré. She can be reached at:

Anna Maria de Alvaré
Lawrence Livermore National Lab
L-303
PO Box 808
Livermore, CA 94450
+1 (415) 422-7007
annamaria@lll-lcc.llnl.gov
uunet!lll-lcc.llnl.gov!annamaria

1003.7 – System Administration

This new working group met as a Birds of a Feather session during the Hawaii meeting. During that session the group convenor outlined the goals and solicited input from the attendees. At a subsequent meeting in Monterey (in conjunction with the USENIX Large System Administration Workshop) the group took the input from that meeting and the work that had been going on off line and began producing a draft document.

So, what is the purpose of this body? To define a portable user interface for System Administration Utilities which would allow users to administer systems in a portable way, and allow developers to build system administration tools on top of consistent underlying commands and libraries. Since the work of this body will overlap with almost every other P1003 working group (and possibly other groups outside of POSIX), coordination is a major part of the standard development effort. Also, because the charter of this group is so broad (what is an administrative tool, anyway?), it is going to take quite a while to complete the standard.

Just to give you a rough idea of what is going to covered by this group, here are some possible areas: machine startup, process management, network, software licensing management, user management, password management, etc... At the meeting in Hawaii it quickly became apparent that the scope of this group is too large to accomplish anything in a reasonable period of time. Some of the time at the Monterey meeting was spent narrowing the scope of the group to a more manageable size. The group tried to identify items which could form a basic set of libraries and commands, and could be finalized in a two to three year time frame. After the initial standard is released, there may be continuing work into areas that the first cut was not able to address.

When I last wrote about this group, I was very critical of its charter and the possibility of it succeeding. I think it only fair to relate that a number of people wrote me and said that I was too judgemental, and that I should take a wait and see attitude. Bowing to the will of the people, I am not going to draw any conclusions about the working group at this time. In the interim, if you want more information, or would like to share your opinions with me, drop me a line.

The USENIX Standards Watchdog Committee contact on 1003.7 is Mark Colburn. He can be reached at:

Mark Colburn
NAPS International
117 Mackubin St., Suite 1
St. Paul, MN 55102
(612) 224-9108
mark@naps.mn.org

1003.8 – Networking Extensions to POSIX

IEEE P1003.8's charter (not yet formally approved by IEEE, but pending) is to help develop an IEEE POSIX networking standard. This was the committee's first formal meeting, and it was devoted mostly to organizational matters, particularly on setting specific technical goals and how to divide the work into subcommittees.

This working group has emerged out of the work done by the /usr/group Technical Committee's subcommittee on networking. Once this committee has been formally formed, the /usr/group networking committee will be considered to merge with the P1003.8 committee, and meet concurrently whenever P1003.8 does. Ultimately, the /usr/group committee is likely to disband completely in favor of P1003.8.

The charter ("project authorization request," or PAR) was reviewed briefly:

SCOPE

1. Define Network Services required by portable applications consistent with existing and emerging standards such as OSI.

2. Define interfaces to the network services defined above, and where possible, language and protocol independent programming interfaces.

3. Identify the requirements for new network services & protocols and liaison with appropriate standards bodies (national and international) and interested organizations when appropriate.

PURPOSE

Define and/or adopt a set of paradigms to permit the implementation of portable applications in a network environment.

Areas to be addressed by this committee include:

1. Interoperability between POSIX applications and non-POSIX applications.

2. Bindings to OSI application layer services.

3. Identification of language requirements not appropriate to applications portability, and liaison with appropriate standards bodies to ensure that action is taken where appropriate.

4. The evaluation and definitions, where require, of binding to lower layer OSI services.

5. The requirements to ensure interoperability among POSIX-based distributed applications and services.

6. Network Services profile definitions for portable applications (POSIX).

Subcommittee Organization

A poll was held to find out what the most important topics were as far as the group was concerned. These turned out to be: process to process communication, directory services, network management, transparent file access, and remote procedure call. Three main subcommittees were formed to look at some of these tasks. Roughly, these committees were "interprocess communication," "remote procedure call," and "transparent file access."

Directory services and network management were recognized as important, but also as cutting across other functional areas. Also, it was noted that there were not enough people in attendance with sufficient expertise in these topics to form a useful body of opinion on proposals in these areas.

Transaction processing was generally felt to be within the domain of the committee, but as a special case of remote procedure call. It was noted that others who were not on the committee may feel otherwise.

The committee split up into subcommittees for a day to refine the definitions of the most important end products identified for the committee to concentrate on.

Specific Technical Goals

The following is a summary of what the committee as a whole agreed on as a result of the input from the individual subcommittees.

- Transparent File Access

It was decided that the products should be client-only interfaces. Three products were identified:

1. Full POSIX-semantic transparent file access interface. This would include previous /usr/group DFS Committee work on DFS (distributed file system).

2. Administrative interface to support (1).

3. Subset-semantic transparent file access interfaces. This could be vendor (e.g., MS-DOS, Apple, etc.) or protocol (e.g., FTAM) specific.

Work items identified so far include:

1. Definition of file operations

2. Liaison to system administration; definitions of transparent file access specific system administrative utilities and/or interfaces

3. Liaison with directory working group

4. "Appropriate approach" to the protocol invention problem

This group expects to finish relatively quickly (6 months or so was the estimate given), because it was felt that a significant amount of the work needed to produce standards in this area is already done by definition (the P1003.1 standard).

- Remote Procedure Call:

The RPC folks apparently did not define their charter so much as identify issues that need to be addressed. The following was their list of issues along with tentative resolutions (if any):

1. Level of service

2. POSIX-to-POSIX versus POSIX-to-other (address POSIX-to-other)

3. Language binding (initial target: C)

4. TP support

5. Connection re-use

6. Call-back/recursion

7. Compiler language

8. Data canonicalization

9. Authentication

10. Our scope versus X.500

11. Standard suite of services need to confer with X3T5 on possible charter issues

12. Idempotency – execute once only guaranteed

13. Long running processes – keepalive & timeouts probably needed

14. Crash recovery

15. Real Time issues – no real time interface

16. Directory services

17. Multiple protocol stacks

The subgroup chose not to identify the next step in the process (apparently meaning that they will wait for proposals).

- Interprocess Communication:

Four products were identified:

1. Simple Protocol-Independent Network Interface

Features:

- Bidirectional byte stream virtual circuits
- Connectionless message exchange
- Read/write support
- Protocol-independent naming
- Asynchronous communication services
- Support for both client and server processes
- POSIX-to-non-POSIX support

Issues:

- How to resolve names in a protocol-independent manner?
- What should the individual functions look like?

2. Simple Structured Data Network Interface

Features:

All of (1), with extensions for data description and machine-independent representation.

- Description of the syntactic structure of the data; when you send the data, you reference the structure.

- Not all functions from (1) may work (such as, read/write)

Issues:

- Structure alternatives: ASN.1, ...

- C data structures (stub compilers)

3. Protocol-Option-Extended Network Interface

Features:

- Provides the ability to access protocol dependent options

- Migration path to potential future protocols

- POSIX-to-any

- Virtual circuits, datagrams

Issues:

- Limited lifespan (?)

- Limited utility

- Usefulness as a migration tool

- Relationship to (1)

4. OSI application level interface

Features:

- A family of interfaces with consistent style and syntax which provides OSI application level services, e.g. FTAM, VT, ACSE, ROSE.

Issues:

- Complexity

- Prioritization (which ones to focus on first)

One issue that surfaced very quickly in the network IPC discussions was the differences and relative merits of sockets and XTI. Some went as far as to say that the differences were significant enough to guarantee "religious wars" over the issue, and/or make any kind of progress impossible in the area of product (3).

Whatever the cause, a majority (8/0/3/3) of participants expressed interest in working on product (1), with products (3) and (4) having a relatively weak level of interest.

The committee will get down to serious business at the next meeting (in January; 5 days). For the next meeting, proposals are being solicited in all areas. The USENIX Standards Watchdog Committee contact on this committee is Stephen Head. He can be reached at:

Stephen Head
Hewlett Packard
263 Mackintosh St.
Fremont, CA 94539
+1 (408) 447-2740
smh@hpda.hp.com
uunet!hpda!smh

That's about it for this quarter. As always, if you have any comments or suggestions, please contact me at:

Shane P. McCarron
NAPS International
Suite 6
117 Mackubin St.
St. Paul, MN 55102
+1 (612) 224-9239
ahby@bungia.mn.org
uunet!bungia.mn.org!ahby

Letter to the Editor

21 December 1988

First, I'd like to make it clear that I support the Association's desire to publish periodically a report on UNIX-related standardization activities. I also respect Shane McCarron and value his opinions on controversial issues in this area. However, I feel that because of the increasingly editorial nature of the Updates, they are no longer serving their intended purpose.

The Updates should report on the accomplishments of the various national and international standards bodies and any other relevant standardization developments or issues that affect the UNIXs community. Controversial issues should be covered impartially, giving both sides equal time and refraining from the pedantic or preachy.

Mr. McCarron's most recent report on the National Institute of Standards and Technology (NIST, formerly National Bureau of Standards) begins with a short update on the POSIX FIPS but quickly becomes a criticism of NIST's practice of writing FIPS before the associated POSIX standard has been approved. He even goes so far as to say "This interesting turn of events proves that in their previous heinous acts they were just being nice." This is uncalled-for. NIST's actions are at worst misguided. To his credit, McCarron says NIST is being given the opportunity to rebut these comments. Unfortunately, McCarron presumes himself capable of predicting NIST's response and proceeds to attack the rebuttal before it has even been made.

There are other examples, but I won't bother listing them. My point is that I don't think the Association should continue to fund McCarron if he can't or won't cease using the Updates as a personal soapbox. If the Association values his opinions enough, he should be commissioned to write a separate editorial column.

David Sill
UNIX System Programmer/Analyst
Naval Surface Warfare Center
Dahlgren, VA 22448
dsill@relay-nswc.navy.mil

Further comments (not flames) concerning these reports may be sent to {*usenix,uunet*}*!peter*.

The EUUG

The EUUG and USENIX have agreed to give space in their respective newsletters so that their members can be made more aware of what is happening on the other side of the Atlantic.

Donnalyn Frey has already started to fill her column in the EUUG Newsletter, informing us of what is happening with USENIX.

Since this is the first article to be published in the EUUG column of *;login:*, it occurred to us that there are probably lots of people who don't know exactly what the EUUG is (this is certainly true in Europe, so I think it's a good guess that the same must be true in the US).

So, here is a quick history of the EUUG, and a description of what it is today, and what its objectives are.

History

The EUUG began life not as an European organisation, but simply as a DECUS SIG in England. As time progressed, it became obvious that UNIX could (and did) run on machines other than PDP-11s.

Many of the people interested in UNIX had no relation to DECUS, and many did not want any such relationship. DEC was not entirely happy about supporting a SIG whose primary purpose was promoting the use of a non DEC operating system. It became obvious that the SIG could not continue in its present form, and that a split would be made from DECUS.

The result of this split was the formation of the UNIX User Group (UUG).

This was a radical decision, because without the logistical and financial support that DEC gives to its SIGs people had to start paying membership fees. There were many people who doubted that the UUG would last. In one way they were right, the UUG did not last very long. There were more and more people coming to the meetings from outside the United Kingdom – especially from Holland and Denmark, but also from Germany.

The Dutch members founded their own UNIX group, and it became obvious that other countries were interested in doing likewise. It also became obvious that the UUG could not remain a British organisation, so, at a meeting of the UUG in 1980 at Heriot Watt University (in Scotland) the UUG was transformed into the European Unix User Group (EUUG).

Sometime later, the EUUG had to again change name, following some not too subtle hints from you-know-who, to the European Unix systems User Group. Since the EUUG and its logo were reasonably well known by this time, it was decided to keep the name EUUG, and only mention "systems" on printed documents.

One of the first decisions taken by the newly formed EUUG was the promotion of what were at the time called "local groups." These local groups were the beginnings of the national groups which exist today.

The national groups exist to promote UNIX in their respective countries. This is a task which would be difficult for the EUUG to do, given its limited resources. The national groups operate more or less independently of the EUUG, except that they are all members of the EUUG "federation." The national groups provide services to their members which are specific to their national needs (national language meetings, national product catalogues, working groups, and exhibitions, for example).

The EUUG serves as a cohesive force for these groups, and organises things which are better tackled at a European level than at a national level. Examples are the EUUG conferences, and EUnet, of which more will be said later.

As other European countries formed national user groups, most affiliated with the EUUG; the only notable exceptions were the Swiss, who prefer to maintain their famous neutrality.

The current members are:

AFUU	France
DKUUG	Denmark
EUUG-S	Sweden
FUUG	Finland
GUUG	Germany
IUUG	Ireland
I2U	Italy
NLUUG	Netherlands
NUUG	Norway
UKUUG	United Kingdom
UUGA	Austria
BUUG	Belgium
ICEUUG	Iceland
HUUG	Hungary

In all, there are approximately 4000 members of the EUUG, most of these being corporate or institutional members.

Portugal is in the process of forming its national group, and we expect them to become formally affiliated, along with Yugoslavia and Spain, at the next conference, which is to be held in Brussels.

Probably the most visible parts of the EUUG to its members (and non-members) are our conferences, EUnet, and the EUUG Newsletter.

Conferences

The EUUG holds two conferences per year, one in spring and the other in autumn. These are independent of the conferences and exhibitions which are organised by many of the national groups. The spring conference is usually planned to be somewhat larger than that of the autumn, and to be of a more "commercial" nature, with, wherever appropriate, an associated exhibition. These conferences are usually organised in collaboration with the national group. It is the involvement of the national group which gives each conference its own specific "flavour," and it is the part of the EUUG to put its own brand on the conference, in the form of structure and organisation. Thus, our conferences tend to be predictable in that they always have the same sort of format and level of content, but also different with the atmosphere provided by the national group.

Just before the conference, we run a series of tutorials. These are somewhat similar to the USENIX tutorials, offering everything from introductory courses to advanced and highly technical tutorials on some specific aspect of UNIX.

EUnet

The network (EUnet) is somewhat more formally organised than than is the case in the USA.

Each country has one national backbone node handling almost all international traffic. There is one international backbone (mcvax) which connects this backbone network with other continents. This structure is very much determined by the high international telecommunications tariffs.

Because of the volume of traffic, the back-bone links are in the process of being replaced with leased lines, to replace the existing X.25 links. Each country manages its own part of the network, some in close collaboration with the national user group, others with somewhat less contact between the two.

The only rules governing who can connect are:

1. They must be a member of the EUUG (or affiliated group).
2. They must refrain from commercial exploitation of the net.
3. They must pay their fair share of the running costs.

As you can imagine, much time and effort is spent working on policies to implement the third point!

The EUUG Newsletter

The EUUGN is published 4 times per year. At present, it seems to have reached a form which satisfies most people, acting as both a newsletter to inform members what each of the national groups is doing, giving reports on conferences (USENIX for example), and as a technical journal.

Although it currently seems to satisfy most people, we are still exploring new methods of improving its acceptability. For example, we are currently experimenting with bilingual articles. People can send articles for publication in their own language. We will translate these, and print the article in double column format, one column being the original text, and the column opposite being the English translation.

We also carry (limited) advertising to help offset the costs of production.

Future input to ;login:

Rather than try to describe each national group, and what it is doing, future articles in this column will come from these groups, each taking its turn to introduce itself, and explain what it is doing.

Contact points

If you have any questions or suggestions, feel free to contact us. Some useful postal and e-mail addresses follow.

Subscriptions and general questions:

EUUG
Owles Hall
Buntingford Herts. SG9 9PL ENGLAND

Tel. +44 763 73039
Fax. +44 763 73255
euug@inset.co.uk

Newsletter (article submission etc):

Alain Williams (EUUGN editor)
Parliament Hill Computers Ltd.
7 Prospect Street
Caversham Berkshire RG4 8JB ENGLAND

addw@phcomp.co.uk

EUnet - General information:

Daniel Karrenburg
dfk@cwi.nl

Philip Peake
EUUG publications executive
philip@axis.fr

EUUG Spring '89 Conference

Brussels, Belgium
April 3-7, 1989

The BUUG will host the Spring '89 European UNIX systems User Group Technical Conference in Brussels. Technical tutorials on UNIX and closely related subjects will be held on Monday and Tuesday, followed by the three day conference with commercial exhibitions.

If you wish to receive a personal copy of further information about this, and future EUUG events, please contact the Secretariat.

Secretariat	*Tutorial Officer*	*Programme Chair*
EUUG	Neil Todd	Prof. Marc Nyssen
Owles Hall	IST	Medical Informaticas Dept.
Owles Lane	60 Albert Court	Vrije Universiteit Brussel
Buntingford	Prince Consort Road	Laarbeeklaan 103
Herts, SG9 9PL, UK	London, SW7 2BH, UK	B-1090 Jette Belgium
Phone: +44 763 73039	+44 1 581 8155	+32 2 477 44 24
Fax: +44 763 73255	+44 1 581 5147	+32 2 477 40 00
Telex:	928476 ISTECH G	
Email: euug@inset.uucp	neil@ist.co.uk	marc@minf.vub.uucp

Future Events

Workshop on Software Management New Orleans, Apr. 3-4, 1989

See page 3.

EUUG Spring Conference Brussels, Apr. 3-7, 1989

See page 47.

Workshop on UNIX Transaction Processing, Pittsburgh, May 1-2, 1989

See page 4.

USENIX 1989 Summer Conference and Exhibition, Baltimore, Jun. 12-16, 1989

See page 5.

Distributed Processing Workshop Fort Lauderdale, Oct., 5-6, 1989

Graphics Workshop V, Monterey, Nov. 16-17, 1989

Long-term USENIX & EUUG Schedule

Sep 18-22 '89	Vienna, Austria
Jan 22-26 '90	Omni Shoreham, Washington, DC
Apr 23-27 '90	Munich, W. Germany
Jun 11-15 '90	Marriott Hotel, Anaheim
Jan 21-25 '91	Grand Kempinski, Dallas
Jun 10-14 '91	Opryland, Nashville
Jan 20-24 '92	Hilton Square, San Francisco
Jun 8-12 '92	Marriott, San Antonio

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. Payment 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.

We hope to have EUUG tapes and conference proceedings available shortly.

Conference and Workshop Proceedings

Meeting	Location	Date	Price	Overseas Air
Large Installation Systems Admin. Workshop	Monterey	Nov. '88	$ 8	$ 7
C++ Conference	Denver	Oct. '88	30	20
UNIX and Supercomputers Workshop	Pittsburgh	Sep. '88	20	15
UNIX Security Workshop	Portland	Aug. '88	5	7
USENIX Conference	San Francisco	Jun. '88	20	20
C++ Workshop	Santa Fe	Nov. '87	30	20
Graphics Workshop IV	Cambridge	Oct. '87	10	15
USENIX Conference	Washington DC	Jan. '87	10	20
Graphics Workshop III	Monterey	Dec. '86	10	15

EUUG Proceedings for Spring 1988 (London) and Fall 1988 (Portugal) are available in limited numbers to North American customers at $40 per copy.

Long-Term Calendar of UNIX Events†

1989 Jan 9-13	IEEE 1003	Embassy Suites, Ft. Lauderdale, FL
1989 Jan 17	Terminal Int. Ext. and Net. Serv.	NIST; MD
1989 Jan 30-Feb 3	USENIX	Town and Country, San Diego, CA
1989 Feb	UNIX in Government	Ottawa, Ont.
1989 Feb 28-Mar 3	UNIX Convention	AFUU; Paris, France
1989 Feb 28-Mar 3	UniForum	Moscone Center, San Francisco, CA
1989 Apr 3-4	*Software Management Workshop	New Orleans Hilton, New Orleans, LA
1989 Apr 3-7	EUUG	Palais des Congres, Brussels, Belgium
1989 Apr 10-11	ANSI X3J11	Phoenix, AZ
1989 Apr 24-28	IEEE 1003	Minneapolis-St. Paul, MN
1989 May 1-2	*Transaction Processing Workshop	Pittsburgh Hilton, Pittsburgh, PA
1989 May 8-12	DECUS	Atlanta, GA
1989 May 14-16	AMIX	Israel
1989 May 16	POSIX Application Workshop	NIST; MD
1989 May	UNIX 8x/etc	/usr/group/cdn; Toronto, Ont.
1989 Jun	NZSUGI	New Zealand
1989 Jun 12-16	USENIX	Hyatt Regency, Baltimore, MD
1989 Jul	JUS 13	Toyko, Japan
1989 Jul 10-14	IEEE 1003	San Francisco, CA
1989 Sep	*Large Systems Admin. Workshop	?
1989 Sep 18-22	EUUG	Vienna, Austria
1989 Oct 5-6	*Distributed Systems Workshop	Marriott Marina, Ft. Lauderdale, FL
1989 Oct 16-20	IEEE 1003	Brussels (or Amsterdam)?
1989 Nov 1-3	UNIX Expo	New York, NY
1989 Nov 6-10	DECUS	Anaheim, CA
1989 Nov 16-17	*Graphics Workshop V	DoubleTree Inn, Monterey, CA
1989 Nov	JUS 14	Osaka or Kobe, Japan
1989 Dec	JUS UNIX Fair	Toyko, Japan
1990 Jan 22-26	USENIX	Omni Shoreham, Washington, DC
1990 Jan 23-26	UniForum	Washington Hilton, Washington, DC
1990 Jan 29	IEEE 1003	New Orleans, LA
1990 Feb	UNIX in Government	Ottawa, Ont.
1990 Apr	IEEE 1003	Montreal, Que.
1990 Apr 23-27	EUUG	Munich, Germany (tentative)
1990 May 7-11	DECUS	New Orleans, LA
1990 May	UNIX 8x/etc	/usr/group/cdn; Toronto, Ont.
1990 Jun 11-15	USENIX	Marriott Hotel, Anaheim, CA
1990 Autumn	EUUG	south of France
1991 Jan 21-25	USENIX	Grand Kempinski, Dallas, TX
1991 Jan 22-25	UniForum	Infomart, Dallas, TX
1991 Jun 10-14	USENIX	Opryland, Nashville, TN

† Partly plagiarized from John S. Quarterman by PHS.

* USENIX Workshops

Large Systems Administration Workshop

There will be a third Large Systems Administration Workshop, most likely in early September. It will again be chaired by Alix Vasilatos, *uunet!osf.org!alix*. A full announcement will appear in the next *;login:* and on *comp.org.usenix*.

–PHS

New Release of 2.10 BSD Available

The second release of 2.10BSD is finally available! It has been designated 2.10.1. Although the changes are fairly simple to describe, they cover large portions of the distribution. Most will not be visible to either users or administrators; specifically, no recompilation is necessary. Administrators should be aware that the 4.3BSD disk quota system is now available. Due to address space considerations, however, it is expensive to run. Also, the source for the on-line manual pages has been rearranged as per the 4.3BSD-tahoe release.

The major change, and the reason for the second release, is an extensive reworking of the kernel to move the networking into supervisor space. This move eliminated most, if not all, of the instabilities seen in the original networking provided with 2.10BSD; it also doubled the speed of, for example, file transfer. As encouragement to sites that encountered difficulties in using the networking in the first release, or encounter difficulties in this release, we have beta sites that have been running for months without crashing, as well as sites with fifty nodes. We are, however, still suspicious of the DEQNA driver...

In application land, many missing pieces of the 4BSD distribution have been added, most notably the FORTRAN compiler and library and the line printer sub-system. Many other programs have had minor (and not-so-minor) fixes applied.

Keith Bostic
Casey Leedom

Because the changes to the kernel are major, no "upgrade" tape will be available. 2.10.1 BSD is only available as source, to appropriate licensees of V7, System III, System V, or 2.9BSD. The cost is $200, prepaid.

The release consists 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.

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

2.10BSD
USENIX Association
PO 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,seismo}!keith
keith@okeeffe.berkeley.edu

+1 415 642-4948

NOTE: There are a few copies of 2.9BSD available. If you do not have split I&D and want to run UNIX on your PDP-11/x, write the USENIX office.

– PHS

AUUG Management Committee Meeting

28th October, 1988

MINUTES

The meeting opened at 10:12 with the following committee members present: President Greg Rose (GR) in the chair, Secretary Tim Roper (TR), Frank Crawford (FC), Tim Segall (TS) and Rich Burridge (RB). Also present was the AUUGN Editor John Carey (JC). The Treasurer Michael Tuke (MT) and Chris Maltby (CM) arrived later.

1. Apologies
   GR advised that CM would be arriving late.

2. Minutes of Last Meeting (12th September, 1988)
   There were no amendments.

   Moved TS/RB **That the minutes of the previous meeting be accepted.** Carried unanimously.

3. Business Arising from Minutes
   (a) Re Item 7, MT reported that the donation to Wollongong Hospital had not yet been made but would be as soon as possible.

   (b) TR reported that he had written to Ken Preiss of **UNIX People** in accordance with the last motion in Item 15.

   (c) GR reported that the **Informix** licence mentioned in Item 15 had not yet been received.

   (d) Re Item 20(b) TR reported that he had filed four applications for registration of a trademark at a cost of $450, being the word **AUUGN**, the logo incorporating a map of Australia and the letters **A U U G** in four vertical stripes, and the word **AUUGN** (in two categories).

   (e) Re Item 20(c) TR reported that he had commissioned a printer to produce the common seal and design and print stationery. He presented proofs of the artwork and a quote for estimated quantities. There was some discussion of prices, colours and designs.

   Moved FC/CM **That the Secretary be authorised to spend up to $1500 on stationery.** Carried.

4. President's Report
   GR reported that there was no progress on the membership database design. He reported receiving good reports on AUUG88 from speakers, exhibitors and members. There was discussion on how to publicise conferences better.

   Moved CM/RB **That the President's report be accepted.** Carried.

5. There was no item 5 on the agenda.

6. Secretary's Report
   TR reported that David Purdue (DP) had offered to arrange a

bulk purchase of *Nutshell Handbooks* from **O'Reilly and Associates.** The discount offered by the publisher and bulk freight rates mean that the price to members of a book should be approximately the same as if they bought it retail in the USA.

There was some discussion of whether sale should be to members only, or to non-members at a higher price.

Moved CM/FC **That the committee approved in principle the proposal outlined by the Secretary for AUUG to make a bulk purchase of Nutshell Handbooks for resale, and in particular that: (a) sale should be to members only for the time being (b) pricing should be based on cost recovery plus 10% to cover unforeseen costs and unsold stock (c) that we should order approximately double the initial quantity requested (d) that terms should be strictly payment in advance of delivery (by AUUG to members) except for Institutional members from whom purchase orders would be accepted (e) an expenditure of up to $5000 is authorised.** Carried.

The secretary tabled correspondence:

(a) to the **Registrar of Trademarks** on 4/10/88 accompanying the applications for registration of trademarks; from the **Assistant Registrar, Trade Mark Operations,** on 18/10/88 acknowledging the applications and advising of a 19 month delay in examining applications.

(b) from Rex di Bona, winner of the AUUG88 Student Prize, on 4/10/88 documenting his expenses incurred in attending AUUG88. (Payment was mailed on 21/10/88).

(c) from **Sugar Research Institute** on 7/9/88 enquiring about availability of back issues of AUUGN. (A catalogue was sent to them on 18/10/88).

(d) from **K. Svendsen** on 10/8/88, **Datamatics Consultants Ltd** of Bombay, India on 2/9/88 and **Department of Defence** on 23/9/88 enquiring about the activities and membership of the group. (Flyer and forms sent.)

(e) from **/usr/group** dated 5/7/88, postmarked 7/7/88, received 5/10/88 (apparently sent by surface mail) regarding the group's entry in their *UNIX Products Directory.* (CM reported that this had already been taken care of.)

(f) from **Monash University** accompanying payment for Institutional membership and requesting acknowledgment. (A receipt had been posted.)

(g) from **Stephen Moore** to **John Lions** dated September 9, 2988 (sic.), claiming inaccuracies in the President's Message in the AUUG88 brochure and in particular claiming that the ComputerWorlds were better; a copy of **John Lion's** reply of 21/9/88 which pointed out that there was no such inaccuracy as the comparison was between AUUG88 and other conferences and exhibitions organised by AUUG, not with events organised by others.

(h) from **Bendigo College of Advanced Education** on 7/9/88 enquiring about the renewal date of its subscription;

replies of 28/9/88 and 29/9/88.

(i) to all members of the Management Committee and to the AUUGN Editor (incorrectly dated 28/8/88) giving notice of this meeting.

(j) to **Labtam Limited** expressing thanks for its support of the AUUG88 Programme Committee Chair.

(k) to the AUUG88 guest speakers **Michael Lesk, Mike Karels** and **John Mashey** on 28/9/88 requesting details of their expenses so that reimbursement could be made.

(l) to **Bennett-Ebsco Subscription Services** on 28/9/88 advising of the current rate for subscription to AUUGN.

(m) to **UNIX People** on 28/9/88 as per Item 3(b) above.

(n) from **H M Bates Australia Pty Limited** on 13/9/88 accompanying the receipted invoice for AUUG88 insurance.

(o) from **IDC Australia Pty Limited** on 12/5/88 and **Department of Primary Industries** on 19/11/87 regarding their lapsed subscriptions; replies of 28/9/88.

(p) from **Australian Exhibition Services Pty Ltd** on 26/10/88: see Item 15 below.

TR reported that he had had the post office box redirected to his home address and that he was handling valid membership applications by passing the forms onto Robert Elz (KRE) for entry into the database and payments onto the Treasurer. KRE appeared to be happy to keep doing this.

Moved RB/FC **That the Secretary's report be accepted.** Carried.

7. There was no item 7 on the agenda.

8. Treasurer's Report

MT reported a cheque account balance of $12,361.02 and term deposits of $27,000. No money had been received from ACMS for AUUG88 yet as they were still chasing late payments. MT agreed to follow up the matter of an interim payment. MT outlined a budget for 1988/89 based on the expenditure in 1987/88 with adjustments for increased costs, membership receipts and planned activities. There was discussion and general agreement to some amendments.

Moved TR/TS **That the budget as amended be redrafted and incorporated in the minutes.** Carried.
[*This was not available at the time this draft of the minutes was distributed.*]

The Chair made a review of the budget an item for the next meeting.

Moved FC/TS **That the Treasurer's report be accepted.** Carried.

9. AUUGN Editor's Report

FC reported that order forms for *Computing Systems* had been received from **USENIX** and that they would be mailed with the next issue of AUUGN. He had re-established communications with the **EUUG** newsletter editor and they were exchanging newsletters once again. The AUUG88 issue of AUUGN (Volume 9 Number 4) had 700 copies printed with approximately 300 being distributed at the conference and 215 about to be mailed to ordinary members who did not attend AUUG88, to Institutional members and to library subscribers. JC and TR have discussed redesiging the cover of AUUGN. JC pointed out the different binding used for Vol 9 No 4 and that it had the advantage that the spine could be printed with the issue details. It was agreed that this would be a good thing and worth a slight increase in cost. JC reported that the advertisement from **Sequel Education Services** on the back cover of Vol 9 No 4 had been charged at twice the normal page rate but that the expected repeat business from Sequel had not been forthcoming. It was agreed that the back cover should be kept available for advertising. JC was pleased that the print run for Vol 9 No 5 had passed 300 and repeated the suggestion that if AUUGN subscriptions were on a fixed period basis members would be more likely to renew on time or at all. There was general discussion of whether the conversion effort would exceed the saved effort. TR asked for the current systems to be retained pending a new membership database and/or the generation of reminder notices. JC and TR to work on a membership drive.

Moved CM/FC **That the Editor's report be accepted.** Carried.

10. AUUGN Sub-Editors
There was discussion of the loose arrangement with Dave Horsfall (DH) to sub-edit a *USENET* section in AUUGN. JC reported that DH had not produced anything for Vol 9 No 5. It was agreed that any arrangement with DH should be terminated and that the Secretary should write him a letter to that effect. There was some discussion of alternatives. RB agreed to prepare something for Vol 9 No 6.

Moved CM/TS **That expenditure of up to $200 on a new cover design for AUUGN be authorised.** Carried.

At 12:30 the Chair adjourned the meeting for lunch. The meeting resumed at 14:08.

11. 1988 Conference and Exhibition (post mortem)
It was agreed that this had been dealt with under Item 4.

12. 1989 Summer Meetings
GR reported that he and Ken Thompson had attended a meeting of WAUG in Perth. This meeting had been organised in only a few days and consisted of lunch plus a three hour session of speakers. Approximately 120 people had attended. This was a good effort. GR understood that a formal petition would soon be received from WAUG to form a Chapter of AUUG.

It was agreed that AUUG would pay for a speaker from overseas to fly to Australia and to all centres holding Summer Meetings. Also, if suitable local speakers were available they could be flown to one or more meetings.

GR was working on someone to organise a NSW/ACT meeting. TR to contact potential organisers of VIC and QLD meetings.

It was agreed that a condition on AUUG funding for these meetings should be that AUUG members receive at least the same benefits as other attendees.

Extensive discussion followed about suitable speakers to invite, both for the 1989 Summer Meetings and the 1989 Winter Conference and Exhibition. A list of names of some thirty individuals and organisations was made. Three preferences were agreed for the Summer Meetings and six for AUUG89. TR is to write to them in turn.

A list of potential organisers in the following areas was drawn up: Perth, NSW/ACT, VIC, Brisbane, SA, Tas, NT, North QLD. GR and TR is to contact them.

13. <u>1989 Winter Conference and Exhibition</u>
The question of speakers had been covered in Item 12. RB and FC were invited and offered to co-ordinate AUUG89. TR to prepare a suggested timetable and to contact potential Programme Committee Chairs. GR to someone who could possibly organise publicity. TR to contact ACMS about alternative financial arrangements for AUUG89.

MT then received and reported a telephone call from Wael Foda of ACMS advising that about $24,000 was still outstanding and that payment to AUUG of about $12,000 could be expected.

There was general discusison of tutorials for AUUG89. It was agreed that there be two morning and two afternoon tutorials in each case with one being for novices and one advanced. Costs should be about $50 for one tutorial and $100 for two tutorials and lunch. GR agreed to organise the tutorials.

14. <u>Secretarial Assistance (cont.)</u>
GR reported that there had been no action on setting up the new membership database. He suggested that the current database should be left with KRE for the time being.

15. <u>PC89 Proposal</u>
The committee discussed the letter of proposal from the organisers of PC89 (tabled in Item 6 above) inviting AUUG to endorse the UNIX feature at their OFFICE TECHNOLOGY 89 exhibition. It resolved not to accept the invitation and that the Secretary should reply notifying them of that decision.

16. <u>Benefits for Institutional Members</u>
There was general discussion of benefits for Institutional members. It was decided to order copies of the **/usr/group** *UNIX Product Directory* for Institutional members in addition to the copies of *Computing Systems* already approved and to publish the list of financial Institutional members in each issue of AUUGN. It was further decided that there were now sufficient benefits in place.

17. <u>Constitutional Changes</u>
No submission having been received from KRE, this item was defered until the next meeting.

18. Other Business

(a) There was general discussion of ACSnet.

(b) It was agreed in principle that financial members should be issued with membership cards annually. Possible formats were discussed; photographs were suggested but rejected. GR offered to investigate possible benefits such as discounts to be obtained from having such a card.

(c) GR posed a question raised by WAUG namely whether its members could join AUUG at a reduced cost which entitled them to normal membership benefits except for AUUGN. TR pointed out that official notices to members were normally distributed by publishing them in AUUGN and it was therefore necessary for all members to receive it. It was agreed that this meant that the answer was no.

(d) JC suggested that promotion of AUUG at other computer exhibitions would attract members. It was agreed that the cost of this would be prohibitive. However we should enlist the aid of Institutional members to distribute membership information in the course of their business including exhibitions. Printing a glossy handout was discussed; this would be expensive. RB offered to spruce up the current flyer; TR will send him the source file.

19. Next Meeting

The next meeting will be on 3rd February 1989 in Melbourne at a venue to be decided. The meeting was adjourned at 16:45.

AUUGN Back Issues

Here are the details of back issues of which we still hold copies. All prices are in Australian dollars and include surface mail within Australia. For overseas surface mail add $2 per copy and for overseas airmail add $10 per copy.

pre 1984	Vol 1-4	various	$10 per copy
1984	Vol 5	Nos. 2,3,5,6	$10 per copy
		Nos. 1,4	unavailable
1985	Vol 6	Nos. 2,3,4,6	$10 per copy
		No. 1	unavailable
1986	Vol 7	Nos. 1,4-5,6	$10 per copy
		Nos. 2-3	unavailable
			(Note 2-3 and 4-5 are combined issues)
1987	Vol 8	Nos. 1-2,3-4	unavailable
		Nos. 5,6	$10 per copy
1988	Vol 9	Nos. 1,2,3	$10 per copy
		Nos. 4,5	$15 per copy

Please note that we do not accept purchase orders for back issues except from Institutional members. Orders enclosing payment in Australian dollars should be sent to:

AUUG Inc.
Back Issues Department
PO Box 366
Kensington NSW
Australia 2033

THIS PAGE INTENTIONALLY LEFT BLANK

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.

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	$300.00
☐ 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/our ☐ Bankcard ☐ Visa ☐ Mastercard.

Account number: _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _. Expiry date: __/__.

Name on card: ______________________________ Signed: ____________________

Office use only: **Please complete the other side.**

Chq: bank ________ *bsb* ___-___ *a/c* ______________ # ______________

Date: __/__/__ *$* *CC type* ___ *V#* ___________

Who: ____________ *Member#* _________

Please send newsletters to the following addresses:

Name:	....	Phone:	.... (bh)
Address:	....		.... (ah)
	....	Net Address:	....
	....		
	....		
	....		

Name:	....	Phone:	.... (bh)
Address:	....		.... (ah)
	....	Net Address:	....
	....		
	....		
	....		

Write ''unchanged'' if this is a renewal, and details are not to be altered.

Please indicate which Unix licences you hold, and include copies of the title and signature pages of each, if these have not been sent previously.

Note: Re ent licences usally revoke earlier ones, please indicate only licences which are current, and indicate any which have been revoked since your last membership form was submitted.

Note: Most binary licensees will have a System III or System V (of one variant or another) binary licence, even if the system supplied by your vendor is based upon V7 or 4BSD. There is no such thing as a BSD binary licence, and V7 binary licences were very rare, and expensive.

☐ System V.3 source	☐ System V.3 binary
☐ System V.2 source	☐ System V.2 binary
☐ System V source	☐ System V binary
☐ System III source	☐ System III binary
☐ 4.2 or 4.3 BSD source	
☐ 4.1 BSD source	
☐ V7 source	

☐ Other *(Indicate which)* ....

AUUG

Application for Newsletter Subscription
Australian UNIX* systems Users' Group.

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

Non members who wish to apply for a subscription to the Australian UNIX systems User Group Newsletter, or members who desire additional subscriptions, should complete this form and return it to:

AUUG Membership Secretary
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.
- Use multiple copies of this form if copies of AUUGN are to be dispatched to differing addresses.

Please *enter / renew* my subscription for the Australian UNIX systems User Group Newsletter, as follows:

Name: .... Phone: .... (bh)

Address: .... (ah)

.... Net Address: ....

....

.... *Write "Unchanged" if address has*

.... *not altered and this is a renewal.*

For each copy requested, I enclose:

☐ Subscription to AUUGN	$ 65.00
☐ International Surface Mail	$ 10.00
☐ International Air Mail	$ 50.00

Copies requested (to above address) ________

Total remitted AUD$________

(cheque, money order, credit card)

☐ Tick this box if you wish your name & address withheld from mailing lists made available to vendors.

Please charge $_____ to my ☐ Bankcard ☐ Visa ☐ Mastercard.

Account number: _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _. Expiry date: __/__.

Name on card: ________________________ Signed: ________________

Office use only:

Chq: bank ________ *bsb* ___-___ *a/c* ______________ *#* ______________

Date: __/__/__ *$* *CC type* ___ *V#* ___________

Who: _____________ *Subscr#* _________

AUUG

Notification of Change of Address
Australian UNIX* systems Users' Group.

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

If you have changed your mailing address, please complete this form, and return it to:

AUUG Membership Secretary
P O 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#* __________