

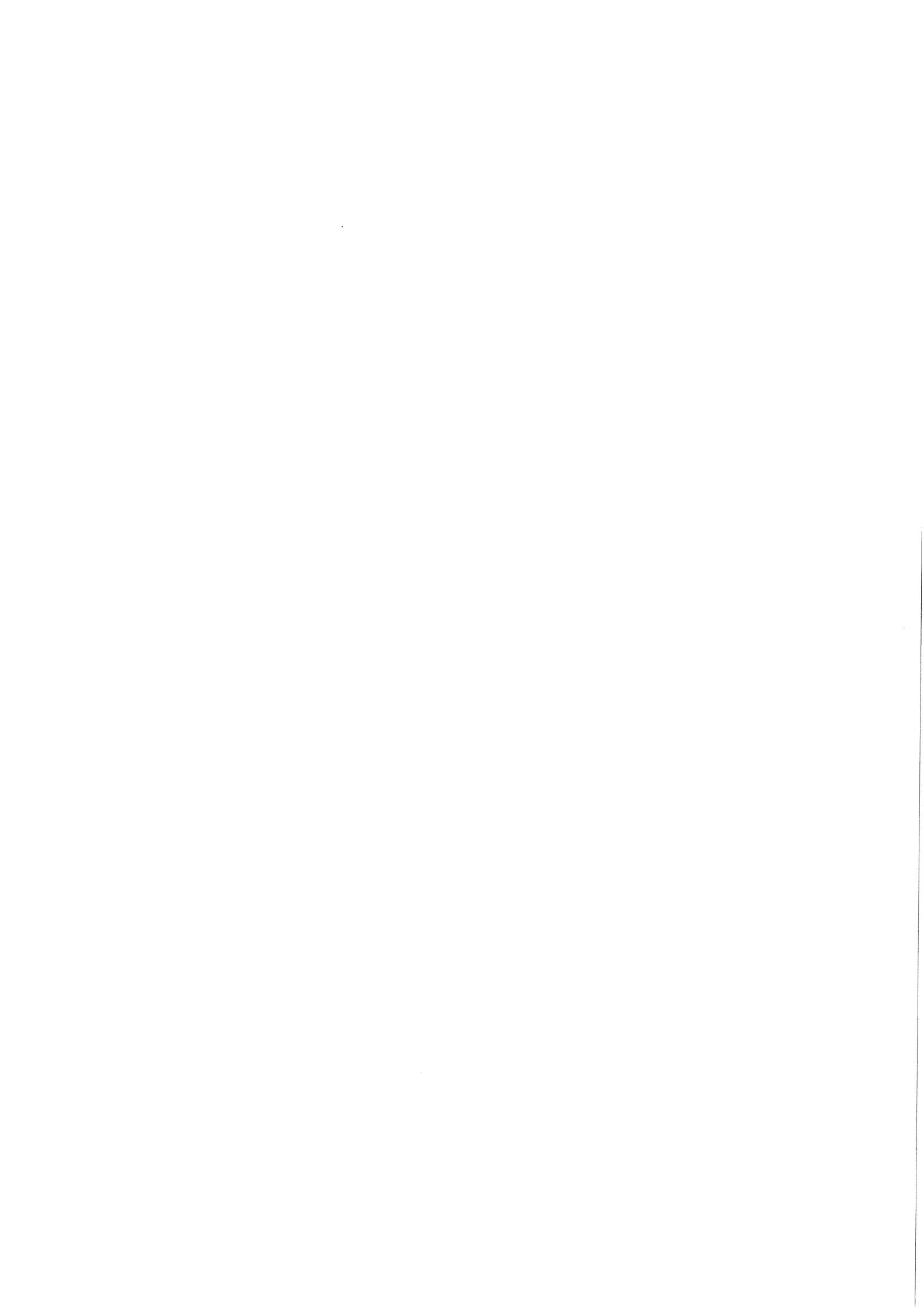
AUUGN

Australian Unix systems

User Group Newsletter

Volume 7

Number 4-5



The Australian UNIX* systems User Group Newsletter

Volume 7 Number 4-5

February 1987

CONTENTS

AUUG General Information	3
Editorial	4
The Claytons UNIX Programmer	6
Network Addressing	13
Small Computers and UNIX	22
Benchmarking Visual Editors	28
Towards a standard programming interface between Graphics Programs and Graphics Devices	30
A cursory view of the state of UNIX on the MV/2000, a Data General Computer	32
Maintaining Geographically Scattered UNIX systems	34
Preface to C++ Overview Paper	39
An Overview of C++	41
Document Production in the UNIX environment	56
Documentor's Workbench on a PostScript Device	71
From the EUUG Newsletter Volume 6 Number 2	89
The Unix Hierarchy	90
RFS Architectual Overview	92
News from Finland - UNIX and the polar bears	103
DKUUG in Paris	105
Abstracts from the Florence Technical Program	107
The Florence Contest	118
From the <i>:login:</i> Newsletter - Volume 11 Number 5	121
Cognito, An Expert System to Give Installation Advice for UNIX 4.3 BSD	122
Access to UNIX Standards	127
Book Review - The UNIX C Shell Field Guide	129
From the <i>:login:</i> Newsletter - Volume 11 Number 6	131
Personalizing the Impersonal	132
Hindsight is 20/20	140

From the <i>login</i> Newsletter - Volume 12 Number 1	141
Call for Papers - Summer 1987 USENIX Conference	142
How To Write a Setuid Program	143
An Overview of the Sprite Project	150
Book Review - The C Programmer's Handbook	155
Standards	156
Letters to the Editor	158
AUUG Membership Categories	177
AUUG Forms	179
AUUG Annual Elections 1987	185
Nomination Form	186

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

* UNIX is a trademark of AT&T Bell Laboratories

AUUG General Information

Memberships and Subscriptions

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

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

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

General Correspondence

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

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

ACSnet: auug@munnari.oz

AUUG Executive

Ken McDonell, President

kenj@moncsbruce.oz
Monash University, Victoria

(Temporary address is kjmcdonell@er.waterloo.cdn)
(University of Waterloo, Canada)

Robert Elz, Secretary

kre@munnari.oz
University of Melbourne, Victoria

Chris Maltby, Treasurer

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

Chris Campbell, Committee Member

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

John Lions, Committee Member

johnl@elecvax.oz
University of New South Wales, N.S.W.

Tim Roper, Committee Member

timr@labtam.oz (This is new)
Labtam Limited, Victoria (This is new)

Lionel Singer, Committee Member

lionel@pta.oz (This does not work)
Lionel Singer Group, N.S.W.

Next AUUG Meeting

The next meeting will be held at NSWIT on the 27th and 28th of August.
Further details will be provided in the next issue.

AUUG General Information

Editorial

Well, judging from the thickness and the content of this issue, the AUUGN has stopped dieting and is starting to look very healthy. Thank you to all those people who responded to my desperate pleas in the newsgroup *aus.auug*, and were concerned by the thinness of the last issue took the time to produce an article. I hope that this enthusiasm is not lost, and people will put the effort into producing a contribution for the next issue.

Some of the papers that appear in this issue were presented at the AUUG Meeting which was held in Adelaide recently. I considered the meeting a great success and that conference organisers should be congratulated. They did a great job despite given very short notice. I enjoyed the proceedings immensely and learnt a great deal.

I suggest if you have not been to AUUG Meeting before, you should plan to attend the next one being held in Sydney. Not only are informative papers presented. It gives you the chance to put faces to names who appear in the local newsgroups, and communicate with you using electronic mail over ACSnet. You can also meet and talk to people who use UNIX from all over AUSTRALIA who you would never get to see in the normal course of events. I am sure you will get as much out of this next meeting as I did with the last meeting.

There are two important issues you should think about over the next few months. They are:-

— AUUG Incorporation.

A postal vote will be held in the next few months.

— Nominating someone for one or the postions of the AUUG Executive.

There is a form in the back of the issue.

Thank you for reading the AUUGN and if you are not a subscriber or a member, I suggest that you fill one of the forms at the back of this issue so as not to miss out on the next issue.

A WARNING to those who are financial members that reminder notices are no longer sent out when your membership expires. You should check the mailing label that came with this newsletter for the expiry date. If it is highlighted you should renew your membership using a form found at the back of the issue.

AUUGN Correspondence

All correspondence regarding the AUUGN should be addressed to:-

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

ACSnet: auugn@monu1.oz

Phone: +61 3 565 4754

(This is new)

Contributions

The Newsletter is published approximately every two months. The deadline for contributions for the next issue is the 17th of April 1987.

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

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

Hardcopy submissions should be on A4 with 35 mm left at the top and bottom so that the AUUGN footers can be pasted on to the page. Small page numbers printed in the footer area would help.

Advertising

Advertisements for the AUUG are welcome. They must be submitted on an A4 page. No partial page advertisements will be accepted. The current rate is AUD\$ 200 dollars per page.

Mailing Lists

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

Disclaimer

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

The Claytons Unix Programmer.

Greg Rose

Softway Pty Ltd

ABSTRACT

Since its quiet beginnings around 1970, UnixTM† has pervaded a very wide range of computer hardware, from small micros to very large mainframes. During this time, there have often been conflicting statements about the future of Unix. This paper attempts to examine the historical forces which have moulded Unix into what it is today, and to extrapolate from that to where Unix may be headed in the relatively near future.

The title of this paper derives from a number of things. Increasingly (and to the good of all in the long run) Unix systems are self maintaining, and on the other hand, programmers do not need to be labelled with their operating system as much as before.

1. Introduction

This paper will attempt to highlight an ongoing phenomenon associated with Unix and its derivatives, lookalikes, workalikes and emulations.

The first few years of the lifetime of Unix were spent in an environment which is significantly different to any common commercial (or even educational or scientific) computing environment. After the first explosive wave of Unix systems installed in Universities, the emphasis changed, and then changed again in an attempt to meet true commercial needs when AT&T decided that there was money to be made.

Despite this, it is only now that Unix is really on the way to achieving this goal, and AT&T have not done all of the work.

2. Past perceptions of Unix

2.1 1970-1974: the early years.

The initial design goals of Unix were very easy to state, and bore no relationship to

TM UNIX is a registered trademark of AT&T.

† In this paper, the word Unix is used to denote an operating system which is intended to be quite compatible to UNIX, and the term includes, for example, XENIX (a trademark of Microsoft Corp), and lookalikes such as Idris and HP/UX. It is the philosophy that counts.

most of the current uses of Unix based computer systems. These were the twin goals of:

- a. supporting efficient program development in an interactive environment, and
- b. Running games programs such as *Space War*.

These goals appeared in reverse order, as Ken Thompson found (while developing a Space War game on a disused PDP-7) that the development environment utilising a batch mode cross assembler, was not good enough. Unix was conceived to be a usable development environment, capable of supporting itself.

The timesharing environment available then was mostly slow, hard copy terminals. Many of these were Teletype ASR33s, much like the old telex machines, with plunger-like keys. Any amount of typing over and above the minimum necessary was unacceptable in this environment, as was any redundant output information. These aspects account for the terseness of the program names and the almost total lack of headings and formatting on the output of standard programs. (Note that the programming approach much applauded in the book "*Software Tools*" by Kernighan and Plauger, Prentice Hall, grew out of the fact that most programs produced *exactly* what was required as output, without frills, and that this was largely what was needed as input to other programs.)

This stage of the development of Unix is the part which has engendered the greatest criticism of Unix, that of its "user unfriendliness". This was unjustified even at the time, as the developers recognised that the environment they utilised was not necessarily one which all people would like. In the paper which announced the existence of Unix¹ this issue was clearly addressed:

"It [*Unix*] offers a number of features seldom found in even larger operating systems, including [*a*] system command language selectable on a per-user basis."

2.2 1975-1980: *The explosion years*.

After the release of the paper mentioned above, a huge number of tertiary institutions around the world accepted Unix as a teaching vehicle, and students began to know it. This was the time when different versions of Unix became available, many enhancements were added, user groups were formed, and the whole Unix scene became somewhat "religious". In this time, there were many arguments about the user interface of Unix, and the world seemed to divide into three classes:

- a. Those who liked the standard shell and command names, and said so.

1. Dennis M Ritchie and Ken Thompson, "*The UNIX Time Sharing System*", Communications of the ACM, Volume 17, Number 7 (July 1974) pp. 365-375.

- b. Those who thought the standard interface to be too terse and cryptic, and said so.
- c. Those who recognised that (for many purposes) the standard interface was too terse and cryptic, and who knew that they could change it, but who didn't want to change it, and didn't bother talking about it.

The problem caused by these groupings was that neither of the vocal groups remembered *that the interface could easily be changed*, while the silent group typically consisted of people who had no incentive to change it, as commercial uses of Unix were still relatively rare.

2.3 1980-1986: the commercial reality years.

This period saw a number of very important developments of Unix for the commercial world:²

- a. Interactive Systems announced supported binary licences for Unix.
- b. Onyx released a Z8000 based multiuser computer running Unix.
- c. AT&T recognised Unix as a product, and announced System III, and then System V - at commercial prices.
- d. The Motorola 68000 family made real big machine performance available.
- e. A number of large machines were built by startup companies, running only UNIX.
- f. Many machines started being shipped with menu systems for system administration and user applications.

The situation with Unix developed into one where the name became of major importance to the computer manufacturers, for two reasons. Firstly, for small manufacturers, the only operating system available to them (unless they employed huge numbers of people to create YAPOS³) was Unix. Secondly, Unix formed the only platform for comparison of many machines, so large sales and tenders often specified its availability, forcing even those manufacturers with proprietary operating systems to make Unix available.

3. The current perception of Unix

There is a dichotomy in the perceptions of Unix in the commercial press. About half of the articles (pertaining to Unix of course) state that Unix "will never make it". The other half seem to implicitly assume that Unix "has already made it". What is the reason for this apparent confusion?

2. I have listed these in an order that is vaguely chronological, based on my quite fallible memory; please forgive anything out of order. Also, many other events may be just as significant, these are just some of the ones I feel are important.

3. Yet Another Proprietary Operating System. YAPOS is a trademark of Softway Pty Ltd.

The real answer is that *both are true*. Unix is in control of an overwhelming number of computers being used in office automation and accounting environments. It has "made it" in the sense that a very large and generally happy set of end users are using it.

The contradiction arises from the fact that almost all such users have never heard of Unix! They do not communicate to an "unfriendly" shell in terse and cryptic commands, rather with some applications packages utilising "friendly" menus or whatever.⁴ So in the sense that MS/DOS⁵ has "made it", and huge numbers of people with PCs know the command syntax, Unix has not, and (hopefully) never will.

4. Needs of the computer-using community

There are a number of needs in the computer using community that are of relevance to this discussion. These are:

- a. Application software which is versatile, widely available, and good value.
- b. Good interfaces for the inexperienced and non-technical user.
- c. A reliable upgrade path.
- d. Support in the availability of training and technical expertise.

5. How Unix will meet these needs

Unix addresses these needs very well, which accounts for the success that it does enjoy in the commercial marketplace.

Because of the excellent program development environment, and the fact that applications once developed enjoy a high degree of portability, there is a great range of software available. The Unix programming environment ensures that much of this software is of high quality and versatility. Note that these issues are mostly important to the developers, and not the commercial users, but it is that much larger segment who profit.

There is no longer any excuse for the commercial user to be exposed to a bad or merely inappropriate interface. Thompson's statement of 1974 (above) is even more valid today. If a single integrated package is in use, its interface should be adequate; if not, there are almost certainly other packages which are, and that one deserves to lose. If the user must communicate with multiple (unintegrated) packages, a simple menu system can be written as a front end, using shell scripts.⁶

4. I personally don't believe that menus are particularly friendly, but I don't want to get into that argument here.
5. also a trademark of Microsoft Corp
6. As a giveaway, Appendix A presents a shell script which interprets menus stored in text files. There is no excuse any more!

The fact that Unix is portable itself, and allows enormous amounts of software to be made available on new hardware at quite low cost, is one of the major attractions of Unix for computer manufacturers. This just happens to ensure the existence of an upgrade path for the commercial user, either within the same manufacturers' line, or outside it.

There are now many courses to teach about Unix, and many graduates from Universities who know enough about it, to provide support at any desired level.

6. Conclusion: The Claytons Unix Programmer

Unfortunately, computer operating systems are fairly tenuous objects, especially in the eyes of a lay person, and Unix is especially so. A more concrete example than computer operating systems can help the description.

Imagine a customer entering a car sales showroom to buy a car. Somewhere in the conversation, he will ask about the engine. If the salesman replies "This model has a diesel powered steam turbine", an immediate loss of interest would probably result. On the other hand, a response of "mumble mumble N cylinder mumble petrol mumble" allows the conversation to continue. Why is this so?

The huge majority of car buyers are not looking for special attributes which may cause trouble. They know that a car with an internal combustion petrol engine can be serviced by any mechanic, and that fuel is available at a nice "user friendly" petrol pump. Many evening colleges (at least in Sydney) give courses about how to service a car, and how an internal combustion engine works. There is no lack of training or service, and the interface is good.

On the other hand, most car buyers, after checking that the car has an appropriate engine, don't allow that thought to reach their consciousness again.

The parallel is between cars and computers, and car engines and Unix. If a potential computer purchaser asks what operating system the computer runs at all, the answer "Unix" will be satisfactory and soon forgotten. An answer like "YAPOS" will cause some loss of interest. Having established that the computer runs Unix, it is then automatic that a large range of application software is available, that good user interfaces can be provided, that support is available, that training courses can be found.

(It also goes without saying that if the above things are not available for that particular computer, you are dealing with a shark in the car industry sense.)

There are people who are prepared to buy cars with strange engines for their own reasons; they might choose to buy an operating system other than Unix for their own reasons, but they must be prepared to sacrifice the convenience. More importantly, when the application is not a straightforward one, there may be reason to buy a "special purpose" operating system (or engine); by definition, the special purpose applications are a small portion of the market place, and should not be allowed to dictate trends to the rest of the commercial users.

An average petrol station does *not* have a sign saying "we provide reciprocating internal combustion engine service (all you steam turbines go away)". It probably says

"Mechanic on duty". In the near future there will be no Unix programmers, just "Programmers on duty". These unspecialised programmers will be familiar with Unix, MS/Dos and perhaps other systems.

They will be the Unix programmers you have when you are not having a Unix programmer.

Appendix A - a simple menu system for Unix

The following Bourne Shell script is intended to run on just about any Unix system.

```
#!/bin/sh
# shellmenu - interpret simple menus.

if [ $# != 1 ]
then
    echo "usage: $0 menufile" >&2
    exit 1
fi

m='basename $1 | sed 's/_/ /g''
n='wc -l $1 | sed -e 's/^ *// -e 's/ .*//'

while true
do
    tput clear; echo "      $m:\n"
    sed 's/@.*//' <$1 | pr -n -t
    echo '\n      Enter selection: \c'
    pause=false

    read sel || exit 0
    case "$sel" in
    0|quit|QUIT|exit|EXIT)
        com=exit
        ;;
    help|HELP|\?)
        echo "enter a number, or the first letter or word of the choice."
        echo "To exit, type '0' or 'exit'." 
        pause=true com=loop
        ;;
    "")
        echo "Please make a selection."
        pause=true com=loop
        ;;
    [0-9] | [0-9][0-9])
        if [ 1 -le $sel -a $sel -le $n ]
        then
            com='sed -n -e 's/.@// -e "${sel}p"<$1'
        else
            echo "Your selection '$sel' is out of range."
            pause=true com=loop
        fi
        ;;
    *)
        com='grep -i "$sel@" $1 | sed -e 's/.@// -e lq'
        if [ "x$com" = x ]
        then
```

```

        echo "your selection '$sel' could not be matched."
        pause=true com=loop
    fi
    ;;
esac

case "$com" in
loop)      :          ;;
exit)      exit 0      ;;
*)         eval "$com" ;;
esac

if $pause
then
    echo "[type return when ready]"
    read x
fi
done

```

Figure 1. the shellmenu shell script.

There is one line which is System V dependent, that is the one which uses `tput` to clear the screen. This can be replaced with an `echo` of either a hard coded sequence if the site uses all the same terminals, or a lot of blank lines works quite well.

The other vague system dependency is in the use of the `echo` command itself. On System III and beyond, the special construct "`\c`" at the end of the string indicates no trailing newline, and "`\n`" adds an extra newline at that point. On older systems, Xenix and the Berkeley variants, there is a flag "`-n`" which must be given to suppress the newline, and multiple echoes for extra lines. Last but not least, the flag "`-i`" on the `grep` command causes case insensitivity on the System III descended versions, but "`-y`" (y? why? I think this rivals even `grep` itself for crypticism) is used on Berkeley and Xenix. Sorry about that.

The files which this command interprets are very simple to construct. They consist of a number of lines, each of which becomes one item of the menu. Each line has two fields, the first of which is the displayed part of the menu and the other is any single line command for the Bourne shell. These are separated by one or more tab characters (and the program above displays these as "`(T)`" for visibility); multiple tabs can be used to line up the commands visually for ease of editing.

Because the command string is '`eval`'ed within the script, it can pass around variables and do things like changing directory successfully (but the name of the file should then be an absolute pathname). In particular, the environment gives one variable a special meaning; the shell variable `pause` can be set to `true` to make the menu wait for the user to type a newline before clearing the screen and redisplaying the menu.

Special cases yielding help and leaving the menu are provided.

Network Addressing

K.R.Elz & R.J.Kummerfeld

kre@munnari.oz, bob@basser.oz

Recently, we have been the recipients of several questions about naming hosts on networks. Rather than continually repeat the answers to individuals, we decided that some kind of broadcast explanation, in more detail than an individual answer would ever obtain might be a good idea. So...

This is a brief(?) discussion of what's involved in choosing a network name (a host name). The approach is general from an ideal point of view, to choose a name on a particular network various network particular criteria might need to be met, such as maximum lengths, etc. Also, on many networks names are assigned, rather than chosen. On such a network, this may guide the assignor.

In this discussion of what a name is we will not be considering any of the following, even though at some level of abstraction they may be considered to be names.

- a. Routing information. A name specifies *what* an object is, not *where* it is. An address (*where* an object is) needs to be obtainable from a name, and then routing information (how to get there) needs to be able to be obtained from this but that is very much an implementation problem, how it is achieved will depend a lot on the particular network.
- b. Directory services. A name is a precise object. The problem of how to obtain this object given a more vague specification is the role of the directory service. Typically a directory service will return a name.

Having eliminated those two it remains to be stated what a name is. Simply, its the unique specification of some object in the network. That is, given a name, one precise object can be located.

The problem of name allocation is to assign names to all objects in the network in such a way that this condition is met.

Given only this objective, name allocation would not be difficult, serially numbering objects would do. However, additional goals are: that the name allocated have some human relationship with the object described (most people should have some idea what object it represents, perhaps after an initial familiarity period), that names be relatively easy to remember, that they not be too long, and that the named objects have some say in how they are named. Names should also be stable, which is to say that once assigned, they should rarely, if ever, change.

An additional goal often encountered is that the administrative load in assigning names be reasonable.

We are going to initially discuss the familiar rfc819 style of domain mail name. That is the general type of name that is used on ACSnet, and on coloured book networks (as well as by the EAN X.400 network user agents). Later a brief discussion of pure

X.400 naming will be given. Other naming schemes can be invented (uucp has one) but it does not seem fruitful to discuss those here.

An rfc822 mailbox name has two basic components, a **local** part, and a **host** part. The **local** part is something that is interpreted at the address specified by the **host** part. The **local** part of the name is often the login or account name of a user; it is not relevant here.

The **host** part will also usually be used for other, non-mail naming, which makes it even more relevant to consider this part.

The set of all available names is broken into subsets, each of those subsets is known as a **domain**, and is given a name, known as a **domain name** and the naming scheme is known as **domain naming**. Each domain is further split into sub-domains, and these are also given names. This continues as long as the domain remains big enough (in some undefined sense) to require it. The basic idea is to spread the administrative load of assigning names, without forfeiting any of the other objectives. The final **host** name is the concatenation of all the assigned (sub-)domain names, from the smallest to the largest. This can be done with the result written in either order, and of course, because it can be done, it has been done!

The UK Coloured book names, and the names used by the rest of the world differ in the order of significance of this series of sub-domain names. For present purposes this is not important.

The format of the **host** part is specified in rfc819. Basically, it comprises a series of names, separated by dots. This implements this domain scheme - the dot (period) serves simply as a separator between the domain names, it has no other significance.

Any naming scheme needs some authority to govern it. That is, something must prevent two objects having the same name. With domain names, this authority is distributed. A central authority exists only to allocate names at the highest domain level. This authority assigns names to other, lesser authorities, which can then assign names within their assigned domains.

It is not required that any authority be a human, anything that has the capability to assign names without causing any ambiguity can be used. Nor is it required that the authority designate the name to be used, in fact, in almost all cases it is far better if the entity requesting a name suggest the one wanted. The authority need only check that this name is not allocated, and then allow or reject it on that basis. However, especially at the higher levels, the authority should also assure itself that the entity requesting the address can truly claim to represent the object being specified.

Now for an example. Assume that a top level domain for Australia has been allocated. This means that Australia has been handed a set of addresses it can allocate. There is no requirement that the objects addressed be Australian, that they be located in Australia, or in fact, that they have any connection with Australia at all. Its the name space that belongs to Australia.

Now in practice, it would be unusual if a non-Australian object wanted to request a name in the Australian namespace, but it is possible - with one caveat. That is that there is no requirement that Australia register anyone in its namespace (many apply, few are chosen...). Given that some objects will be registered, any criteria at all can be applied to decide which, and what names they should be given.

In the international community, only the ISO and CCITT have any claim to having any international control over networking, and both of those are concentrating on X.400 (aka MOTIS), so neither has any great interest in regulating rfc819 domain names. However, these names exist, and will do so for some time, so someone needs to control their allocation (or really, control allocation of the highest level of domains). The biggest user of rfc819 and in fact, its creator in some sense, is the US Arpanet, so the responsibility of allocating rfc819 top level domains has been delegated by the agreement of everyone concerned to the Arpanet Network Information Center (NIC).

They have allocated one domain to each country, fixing the ISO standard 2 letter country abbreviation as the domain name for each country. The name for Australia is AU. Any addresses ending ".AU" (or starting "AU." in the coloured book scheme) are ones that have been allocated by the Australian name authority (or if not so allocated, are invalid). This naming scheme happens to be the same one being contemplated in the X.400 world, so "AU" will be Australia there too.

How the various countries will allocate their internal namespace, and who in each country will be responsible for this policy cannot be decided by the Arpanet NIC - that is left to the groups in each country who run the networks to work out for themselves.

In Australia there neither is, nor has there ever been, any network administrative structure to claim the AU domain, yet the ability and desire to make good use of it have existed for some time. With that in mind, it was decided to go ahead and apply for the AU domain to be allocated, so the networks in Australia with the ability to use the global mail system would not be held up waiting for someone to set up the right bureaucratic infrastructure.

So, the authority for AU was delegated to Robert Elz, of the University of Melbourne. The policy for allocating names in AU is that the authority requesting the name must represent a substantial community in Australia. Initially, only one such community existed, ACSnet, but as it was unlikely that would always be the case, ACSnet was not simply handed "AU" domain, rather it was given a subdomain of it. This leaves open the possibility that other organisations or people with some influence in the networking area could obtain other subdomains of AU to administer.

At the current time, only one name has been approved in Australia. That one is "OZ". This name has been given to the ACSnet administration. That is, any names ending ".OZ.AU" (starting AU.OZ.) are names that have been allocated by ACSnet.

Here there have been many names allocated. The authority that controls this allocation is the SunIII network code developed at the University of Sydney. The policy in allocating names in OZ is that the object to be named must be connected to ACSnet (using the SunIII code, or some other mechanism).

Names allocated inside OZ may be further subdivided according to the policy of the authority the name was allocated to, and within the guidelines imposed by the SunIII network code when it grants this authority. There are some restrictions here concerned with the implementation of SunIII.

Now to the objective of this entire article. How does someone obtain a network name? Well, all that's required is to find some authority that will register you in their domain. Any authority will do, but it must be one that is willing to register you - that is, you must comply with the rules of that authority.

If you are going to run the SunIII network code, then things are fairly easy, you qualify for a name within OZ, you select the one you want, tell the network code the name you have selected, and then see if it agrees to let you use it or not (it will unless the name exists elsewhere).

But a little more thought is required here for the name choice to be made rationally. First, it makes sense for an institution to obtain a name, and then subdivide that amongst the components of the institution, rather than each component obtaining its own name. This is just a good social policy - the less names taken from a domain, the more are left available for future requestors. Taking more than you need is just greed. You must also comply with the slightly baroque requirements of SunIII, which in some instances run directly counter to the previous point.

Now let us suppose that you are not going to be running SunIII. Here we must examine who you are a little. If you are a small site, a single host, or small (comparatively) network, then it probably makes sense for you to register your name inside OZ, or perhaps some other registered subdomain of AU (when there are more).

If you are a large, multi-organisational, network, then it would probably make sense to obtain a new subdomain of AU - this provides a whole new clean namespace for you to allocate, and you won't be subject to any control from SunIII (remember that subdomains of OZ must comply with all SunIII's rules when allocating names).

Let us assume (hypothetically for a minute) that another subdomain of AU exists, lets call it AC in order to have something concrete.

Now, if you are a site that wants to obtain a name, what do you do? Things are a little more complex now, as instead of just taking a subdomain of OZ because that's all that was available, you can now pick and choose - assuming that both authorities will accept you.

Here it simply doesn't matter which you choose - whichever best suits your needs. What is important is that you choose *one* of them, and NOT obtain names from both. The latter is legal (if the authorities permit it) but can be very confusing.

It should not be important here which authority you choose, though in many cases one of them will be the *obvious* one. Since a name specifies *what* you are, and not *where* you are, having a name allocated by one of these authorities does not constrain which networks you are connected to.

That is, assuming that you obtain a name FOOBAR.AC.AU (or perhaps AU.AC.FOOBAR) there is nothing to stop you connecting that object to ACSnet. SunIII has a few rules that you must follow, one in particular is that you must be internally connected to all other AC.AU (AU.AC) hosts, but that's all.

Similarly, assuming that you have a name FOOBAR.OZ.AU (or AU.OZ.FOOBAR) nothing is going to prevent you connecting to the network which owns the AC domain (assuming that in fact, there is such a network).

It is the job of the various network code to route messages to the objects identified by the various names. We believe that both systems (SUNIII, Coloured book) in use in Australia can adequately cope with this since they use this naming form.

As an absolutely concrete example, The University of Melbourne has been allocated MU.OZ.AU (and has also grabbed for itself a whole swag of other names in OZ.AU - a few of which it was required to take by the SunIII rules, but many more which are the result of simple greed).

This should be its name on all networks. There is NO need for any other names to be allocated to that University, one is quite enough (or should be quite enough).

Now, for something a little different, lets consider the future. It is beyond doubt now that future mail networks will be based on the CCITT X400 standard (or the equivalent OSI MOTIS standards, that is, once they actually become equivalent).

Here things are done a little differently, though many of the principals are the same.

In the X400 system addresses are highly structured. The term *Originator/Recipient Name* or *ORName* is used to refer to a name.

An *ORName* can take a number of different forms, some of which aren't relevant in the current environment since they contain either a *Unique User Agent Identifier* (a unique code for every user on the net) or an X121 address (a terminal address for X25 or Teletex networks). The form that we will almost certainly use in the future has the following parts:

Country Name
Administration Domain Name
Personal Name[*]
Organisation Name[*]
Organisational Unit(s)[*]
Private Domain
Name[*]
Domain Defined Attributes[*]

Country Name This is an ISO code that can either be a number or a two character string. The code for Australia is AU.

Administration Domain Name (ADMD)

This refers to a service provider such as Telecom. It can also be a number or printable string. We don't know Telecom's code yet (if

it has one).

Personal Name This is made up of the following component printable strings: surname, given name, initials, generation qualifier. Only the surname is required, the rest are optional.

Organisation Name, Organisational Unit(s)

These are printable strings.

Private Domain Name (PRMD)

This can be a number or a printable string. For Acsnet it would make sense to choose oz.

Domain Defined Attributes

These are pairs of printable strings — a type string and a value string. This part of the address can be used to convey information that is meaningful to the destination private domain.

The first two components (Country and Admin domain) must be present in all ORNames of this form. At least one of the components marked with [*] must be present in an ORName.

A **printable string** is a string that contains only characters from a subset of ASCII (actually International Alphabet No. 5...). The subset is:

A-Z a-z 0-9 '()' , . / : ? and space

Notice that it doesn't contain at-sign (@). Its also worth noting though that this limited character set is currently subject to review, as it doesn't contain all of the alphabetic characters used in some European countries, and doesn't suit the non Latin alphabet countries at all. It is quite likely to be extended.

It is possible to design a mapping between our current form of names and X400 ORNames. This would allow a user on a "pure" X400 system to send a message into the current WorldNet (Acsnet, Arpanet, Bitnet, Csnet etc). Steve Kille of University College London has proposed a mapping that would be adequate. The basic idea is to use the Domain Defined Attribute to carry an encoded form of our current names. An example name is:

Country name	=	"au"
Admin Domain name	=	"Telecom" (for example)
Private Domain name	=	"oz"
Organisation	=	"su"
Domain Defined Attr	=	"bob"/"basser"

In our current notation, this might be written

bob@basser.su.oz.telecom.au

Or the "telecom" might be elided if it can be inferred from the rest of the name, giving a more familiar

bob@basser.su.oz.au

There is also a reverse mapping that allows an *ORName* to be encoded as an Acsnet (or other) style name. Here is an example:

/C=au/ADMD=Telecom/O=BHP/OU=Steel/PN=J.Smith/@munnari.oz

This message would be sent to munnari and then (via X400 somehow) to J Smith in the Steel department of BHP.

These examples don't really do justice to Steve Kille's proposed mapping. The issues are very complicated and his document (70 pages of it!) tries to cover all the problems.

While this technique will be useful in the medium term it is not a very satisfactory long term solution. In the long term we should move to X400 addresses of the form described earlier.

Whichever mapping is used (which is to say, however names are written down), the name allocation problem is very much as before.

At the top level are countries - the names for those are predetermined in the standard. The designated body in each country (in Australia it will almost certainly be the Australian Standards Association) will then allocate Administrative domains to eligible bodies. In Australia Telecom, and perhaps OTC (maybe even AUSSAT) are likely to be the only Administrative domains allocated. These administrative domains then allocate Private Management Domains according to whatever policy they set, or they can simply allocate Organisation names.

In a PRMD, that PRMD will allocate Organisation names, and the Organisation will allocate Organisation Units, which will allocate Personal names. Domain defined attributes will be of a form specified by the ADMD or PRMD, and will be allocated however is appropriate to that definition.

Thus, the allocation problem is much the same, the details differ, and the notation differs considerably, but that is not of immense consequence here. A mapping can be defined between the two notations, and that will allow addresses allocated in either scheme to be used in the other, so we would expect that addresses allocated now could continue to be used well into the X400 era.

To sum up - name selection is important. It's important that related entities have related names, and that these names remain stable. Changing a network name because a department changes its name or brand of computer, or other things like that should be discouraged at all costs. Its generally much better to have a name that was chosen on what are now obsolete principles, than to go through the headaches involved in altering an existing name. This should make it very clear that choosing a name should be done correctly the first time!

Lastly, some more practical help in deciding what name you should actually choose. In this area, its wise to bear in mind the requirements of X.400, so when the transition occurs, names need not change if at all possible - only the technology that delivers the

mail, and the precise form of the bytes that are delivered.

You are almost certainly going to want to be registered in the Australian domain, that will be common to rfc819 domain names, and to X.400 names. For now, we will forget ADMD's.

We will assume that X.400 will allocate a PRMD of "OZ" to ACSnet, so if you are adding a host to ACSnet, your PRMD will be "OZ", or your address will end in ".OZ.AU" (of which ACSnet uses only the .OZ).

Within the PRMD you should have a domain allocated to your organisation. The name for this should be something fairly short, yet distinctive, and easily associated with the organisation. For example, the University of Melbourne has "MU". "Melbourne" would be too general, and "University-of-Melbourne" or "Melbourne-University" too long. "Melb-Uni" would be a possibility, but "MU" seems to suffice.

Similarly The University of Sydney has "SU", Telecom Research Labs has "TRL" and CSIRO's Division of Maths and Stats has "DMS".

Sometimes its hard to decide whether some fraction of an organisation should be regarded as an organisation itself, or a subdomain of a larger organisation. Here its probably best to look at the overall general control of the network in the organisation - if its distributed among the units and they are reasonably large, then its appropriate that they are each treated as an organisation for network naming purposes, so its probably better for DMS to be a domain of its own, than a sub-domain of "CSIRO".

Within this, there will often be subdivisions of the organisation. Where they exist, each subdivision should have its own domain to administer. The Computer Science department at the University of Melbourne is "CS.MU.OZ.AU".

Typically this is enough levels, there is no need to add extra nodes in the tree to match with internal organisational breakups (so, there are no domains for the School of Mathematical Sciences, of which the Department of Computer Science is a part, nor is there one for the Faculty of Science, of which the School of Mathematical Sciences is a part). Making a fairly bushy domain tree makes names that are short enough for people to remember and use.

On ACSnet, this should usually be the **hierarchy** for your node.

Ideally, this would be all that would be needed, individuals should simply be named within their department. However, in practice it happens that the mail systems that exist often require that mail be addressed to a particular host computer within the department, and not just the department itself.

Similarly, the SunIII code requires that each host have a private name, and most other networks do likewise.

Also, users (and administrators) typically like to have names for their computers, it is easier to say "foobar is broken, stupid machine!" than "the third machine from the left in the second row is broken...", or "the machine with serial number 12345 is

broken”.

Choosing this name is largely up to individual sites, almost anything is possible. One thing to avoid usually is naming computers after their manufacturers. Quite apart from the fact that the manufacturer's name is usually a trade mark, which you probably won't get permission to apply to your particular computer, it also means that if the machine is upgraded to a different model, or an entirely different brand, then its name would no longer be appropriate, even though the usage, user population, etc, is still the same. Simply serially numbering (or lettering) hosts is one technique that works, though it doesn't show much imagination. Picking names that are part of some common series (flowers, animals, actors, ...) can be a good idea.

On SunIII this name will usually be the **hostname**, though in some circumstances, one of the names of the hierarchy might be a better choice if the particular computer is to act as a gateway to the domain. The SunIII **primary domain** is another thing altogether, it must be one in your hierarchy, and that one must be chosen according to SunIII's routing rules. It is really a routing parameter, and shouldn't be related to naming at all. Unfortunately, currently it is.

If you consider your primary affiliation to be with Spearnet, then you will have to follow their rules for name allocation, but we would expect that they will be something rather similar to the ones for SunIII.

If you have a private internal network operating already (perhaps a DECnet, or some other manufacturers proprietary network) then you will might want to consider that network as a domain within your organisation. At Melbourne University, the local DECnet is “DN.MU.OZ” — even though there is no “DN” University Department, and even though the network encompasses organisations that are not part of Melbourne University. This is a good example of names being allocated by an authority other than what might seem to be the obvious one - these organisations have names allocated by the administrators of the DECnet, so they are in the MU domain. Its the authority that allocates your name that supplies the upper levels of the domain string you get.

Within your private network you will, as usual, need to follow whatever rules are applied by that network.

One final word of advice - try to avoid mixing characters that cause confusion when seen together. A name “hell” (that is ‘H’ ‘E’ ‘L’ ‘one’) is almost certain to cause lots of confusion, being interpreted as almost anything but the name you chose. However compelling the reasons for picking a name anything like this, and however obvious it might seem to you, this type of thing is a very good one to avoid from the start.

Small computers and UNIX.[†]

Ross J Hand

ABSTRACT

A description of the design of a computer system based upon the IBM AT model as a replacement for a DEC mini computer in a university environment.

The following may be copyrighted, trademarked or otherwise tied in a legal web. I acknowledge them all here. ACSnet, APC, ATT, AUSAM, CPM, Cyber, DEC, IBM, Intel, MSDOS, NEC, PDP, SCO, SPSS, Venix, Xenix, Zilog and microvax.

Four areas involved in the design of the computer system will be discussed in this article.

The areas of concern are:

- 1 Hardware
- 2 Operating system software
- 3 Application programmes
- 4 User interaction and education

Introduction

The availability of low cost IBM AT compatibles has allowed a re-evaluation of the hardware requirements for small computer systems. While most users of these small computers will use MSDOS as the operating system software and therefore limit themselves to one user per machine there will be users, who for economic reasons will attempt to put these machines to greater use. The move of a section of a School of the UNSW from one off campus location, to another off campus location provided the opportunity to advise on some aspects of a small computer system. The overall requirement was for the replacement of existing facilities, low initial cost, no hardware maintenance contracts and self support of both hardware and software.

The users requirements were:

- A Text processing facilities for the preparation of papers, questionnaires, letters and data.
- B A terminal per user so that there was no queuing or hardware patching
- C Access to the computer facilities on the central campus.
- D Individual small machines so that persons writing thesis could have independence or a personal machine for use at home.

The major requirement of the section was self sufficiency. There would be no full time support for the system from programmers or technicians. The current staff would need to develop sufficient skills and confidence to maintain the system themselves. One solution would be to add to the existing number of MSDOS computers. Access to the central campus was actually access to a machine capable of running SPSS. SPSS is a large statistical package and had recently become available as a programme for MSDOS machines. The section could contain up to sixteen

[†] UNIX is a trademark of Bell Laboratories.

staff so to meet the requirement of a one terminal per user an additional 10 machines would have to purchased. These machines would have to be limited to floppy disk based machines without printers to keep the cost within budget. The advantage would be a large software base, (including SPSS) reliable low maintenance hardware and a little need for local software and administrative expertise. The system could be purchased piecemeal as requirements dictated. It would be a conservative solution, but safe from criticism.

Another solution would be the purchase of a single multiuser computer and a number of terminals. This solution would be more difficult to implement. It would involve the integration of existing single user hardware and software, increased level of skills by some of the users, the need for an administrator to cope with the increased complexity of the operating system and connections to the central campus. The long term benefits of a multiuser system where considered to outweighed the short term problems.

Hardware.

The large number of machines available at very low cost based upon the Intel sixteen bit microprocessors (types 8088, 8086, 80186, 80286 and 80386) makes them attractive as the basis for a computer system. Even though transportable operating system and application software is making the choice of central processor unit (CPU) less important, certain CPU types are favoured by manufacturers of mass produce computer hardware. There is little use of Zilog Z8000 CPU or National Semiconductor NS32016/NS32032 in mass produced machines and so machines of this type where not considered. The popular CPU types are the Intel and Motorola. The Intel CPUs are found in vast numbers in single user MSDOS machines. The Motorola 68000 and it's derivatives, are found in specialise graphics oriented single user machines and in multiuser machines most commonly using the Unix operating system software. The choice of a single user or multiuser system dictates the hardware for that computer system. This polarisation of the massed produced market has led to some misconceptions. One of the most widely held misconceptions was the ability of the Intel CPUs. IBM adopted a low performance implementation of the first Intel sixteen bit CPUs as their standard for single user operation. This may have been to protect their large multiuser machine base. The result, for what ever reason was to impress upon the buying public the single user, single process nature of Intels CPUs. Certainly the use of the eight bit bus version of the sixteen bit CPU, and at a lower clock speed than was possible created a hardware precedence that other manufacturers found difficult to ignore. A very small number of manufacturers had the marketing power to produce machine based upon the Intel CPUs which where not single user, single process. The majority simple produced imitations or clones of the original low performance IBM design relying upon low cost to attract buyers.

The introduction of a the IBM AT computer, with large memory capacity, high CPU performance and hard disk secondary storage as standard did not change the computer buyers attitude. Although there are many implementations of the Unix operating system available for both the IBM AT and the early PC computers they have not been as popular as predicted. The vast majority of these machines still use MSDOS, a single user, single process operating system. Machines of the IBM AT type are usually considered inadequate when a multiuser hardware is required. The popular choice would have been equipment produced by Digital Equipment Corporation (DEC) from their LSI or microvax range or from the numerous manufacturers using the Motorola CPUs. The use of DEC computers has been popular within universities but the cost of hardware maintenance and the high cost of peripheral equipment makes them less attractive. Small departments with limited recurrent funds cannot afford hardware maintenance contracts and their is a lack of technical expertise for local repair. Skills in programming are more generally sought than hardware repair skills.

The most suitable hardware is low in initial cost and sufficiently massed produced to be treated as expendable. The IBM AT design meets these conditions. The many clones provide competition among manufacturers which has driven down the initial purchase price. Hardware maintenance contracts are not needed because of this low price and also because of the numerous suppliers of peripherals for this design. Repair policy has become one of replacement of defective

sub assembly.

The decision was made to use an AT design as the basis for the system. The hardware used was an IBM AT equivalent design by NEC. To the basic machine was added a Maths co-processor (80287), a memory board containing 2 Megabytes of memory (expandable to 3 Megabytes if required) and a microprocessor controlled serial port card. The Maths co-processor and memory card where available from NEC as standard optional items. The extra serial ports where not and when questioned about this the reply was that the two serial ports provided in the basic unit would be satisfactory for most users applications. NEC sold their IBM AT equivalent with MSDOS, no multiuser operating system was available from them. This may change as more of these machines are used in a multiuser role. Companies like NEC may provide Unix software support as a matter of policy.

The choice of the serial port card was the main problem of the hardware design. NEC had no such card and the majority of peripheral suppliers for the AT design could not supply from stock. The most popular expansion card for serial ports consisted of an extension of the standard serial port design to four, eight or sixteen ports. These cards required operating system software for character transmission and reception, thus placing the central CPU under considerable load if many terminals where in active use. They where also very expensive given there complexity, averaging about 20% of the basic computer hardware cost. Alternatives consisted of locally designed and manufactured cards which contain microprocessors to relieve the central CPU of some of the software task involved in serial communication. These are sometimes referred to as intelligent serial cards. Three local manufacturers provided intelligent serial cards for evaluation. The problem with intelligent serial cards was one of integration. The Unix operating system software has to be modified to accept these new devices. All three manufacturers who provided intelligent serial cards provided this integration. This support from the manufacturers, plus the facilities provided by the Unix software made this task less difficult than expected. All three serial cards where considered adequate. All three cards caused the system to crash, and all where consequently improved. Reliability finally decided the card which was used. The high cost of this one component and the lack of choice should change as more manufacturers produce serial expansion cards. This will continue to be the weakest component in the system until then.

This then was the hardware basis for the system. A NEC APC IV with one 40 Megabyte hard disk, a 1.2 Megabyte floppy disk, 1 parallel printer interface and 10 serial ports. The future expansion would include another 8 serial ports. Existing MSDOS machines would used as terminals or as stand alone machines. To provide for external communications two 2400/1200/300 auto answer, auto dial modems were added. This last item proved to be the most unreliable component of the system, requiring more attention than was anticipated. The modems function was critical to the success of the system but the supplier, having oriented his product to the single user market, had little ability to solve the problems that arose, when his product was used in a network role.

Operating system software

The decision to use the hardware in a multiuser role dictated the choice of operating system software. In reality the choice of hardware and operating system software was made concurrently. It was not an objective decision to use Unix although sound reasons for its use do exist. An important reason is the very nature of computing hardware itself. We have available more powerful hardware every year. The limitation of one task and one user per machine is harder to justify as the hardware improves. If users are exposed to the advanced concepts and procedures available through complicated operating system software like Unix the transition is a gradual process. Networking of machines is also important in this education of users. The operating system software must be capable of supporting a sophisticated network, at both the Local Area Network (LAN) and further. Some solutions to networking problems exist in MSDOS but they lack enough generality to allow communication from the same building, to the same city and to international access. Fortunately Unix has such a network in ACSnet and as an added bonus was free for university sites.

There is also the little understood problem, among Unix users, of the problems of sharing peripherals. Spooling programmes for printing are an accepted feature of multiuser operating system software. With a large number of single user machines the printing problem is solved by having one printer per machine or by having a central printing machine and using a LAN or floppy disks for file transfer to that machine. Both solutions are expensive or cumbersome.

There are at least three Unix software products that are available for the AT design. These are Venix V, Xenix V and Microport SV/AT. All three are validated System V implementations of Unix from ATT. All three are binary incompatible with each other and all three have strong and weak areas. The most popular has been Xenix V from SCO/Microsoft. It's advantages are reasonable support in this country, a version for the IBM PC as well as the AT, a large range of user programmes from both SCO/Microsoft and third party suppliers and it is a fairly stable product. Its disadvantages are a high cost, its mixture of Unix System III and Unix System V and its closed nature. It is marketed by Microsoft (of MSDOS fame) as the multiuser version of MSDOS and will remain, given the source code pricing scale, a closed box in this country forever. The other major contender, Microport SV/AT has more potential, especially in a university environment. It was produced with very little change from the original port of System V for the Intel 80286, by some of the software engineers who did the port. Its disadvantages are that it is a less refined product, has virtually no support in this country and at the present has only a few user programmes available. Its advantages are low cost, standardisation with other Unix System V implementations and the potential to be available at very low cost at the binary level to universities. There may also be complete source code available to universities currently holding source licences to Intel's Unix System V. This would have advantages for both university and commercial users in this country. To those who would cry 'train spotter' I would emphasize the generally poor support that binary programmes of any type have in this country. Past experience has shown that simple software faults can be fixed if the source is available and defy repair if source is not. The holding of source for all key programmes is, I argue a necessity not a luxury. The Unix university community have grown accustom to the attitude of fix it now and report it later. The trend towards binary only support in this country of key software like Unix weakens our ability to do even minor software maintenance. I do not advocate the holding of source at every site. Those days are gone. Uniformity of systems demands less access to source. Reliable and efficient maintenance requires either a central, local holding of source code or an extremely efficient and responsive contact with the holders of the source where ever they may be. My experience would indicate the latter currently does not exist, except under expensive software maintenance contracts. The possibility of better long term support made Microport SV/AT the best choice.

Application software.

The largest single use for the system was text processing. Microport SV/AT has nroff, troff and the new device independent troff (ditroff) as standard software packages. The line editor 'ed' and the screen editor 'vi' were also available. The users had expressed interest in continuing the use of an enhanced version of 'ed' called 'e' and 'roff' which is a simpler version of nroff. Both of these programmes had been available on CPM, MSDOS and finally Unix machines in the users previous work area. Since roff was assembler based an alternative was sought. A public domain programme called proff (portable roff) was found. It exceeded the command set of roff and had many of the features of nroff. It also proved extremely portable, and was compiled under both Unix and MSDOS. The local enhanced editor 'e' was also compiled under Microport SV/AT after minor changes to ioctl system calls. Local plotting software recompiled without changes. This left only ACSnet software and some personal software to transport. After defining the system V flag and undefining the AUSAM flags the standard UNSW source of ACSnet compiled within a few hours. A call programme for the modem was installed and with very few software problems, most of them operator induced, the system became a working ACSnet node. A previous attempt to do this using Xenix had required a special version of ACSnet that had been extensively modified.

The standard mail programmes supplied under Microport SV/AT (mail and mailx) were supplemented by a local mailer which recognised network addresses and the network aspect of the

system was operational. To provide for access to the central campus Cyber mainframe and it's SPSS software the UNSW 'submit' programme, after minor changes to functions accessing password structures was compiled and tested. Other local programmes written in C and awk were compiled and tested. No serious problem was found with any of this software. Serious hardware problems with the modems spoiled the success of the installation, but were eventually solved by replacement. The Kermit programmes were installed on the Unix and MSDOS machines. This allowed the MSDOS machines to be used as terminal emulators and allowed for files under MSDOS to be sent to Unix for printing, backup or as input data to SPSS at the central campus computer. The key decision here was that the most used programmes (editing and text processing) would be the same on both Unix and MSDOS.

During this initial period several improvements and shortcomings were noticed in the Microport SV/AT software. Some of the differences from Level 7 Unix were:

- 1 a device driver for the serial ports that allow one modem to be used for networking to other machines and also allowed logins. Internal locking in the device driver prevented interference with each other.
- 2 The line printer programmes, although appearing at first to be unnecessarily complicated, allowed for installation of new printers with varying requirements without the need for programme source. The user has only two programmes to learn as in most Unix systems. Even though there are more programmes and options for system administrators, few problems have arisen. The print spooler programme has no way of rejecting large print jobs and will require modification to allow recording of paper usage. These features are available under AUSAM systems and are missed. Spooling binary files leads to wasted paper and the annoyance of other users.
- 3 The need for source of the getty programme has been eliminated by the gettydefs file. This text file allows for reconfiguration of terminal parameters. Unfortunately there is no method of determining terminal type in this file and there was no equivalent of the programme 'tset' which is available under Xenix.
- 4 The 'inittab' file, which controls the action of 'init', 'startup' and 'shutdown' programmes is certainly a departure from the simple approach of the level 7 'init'. It's benefit or otherwise has yet to be determined. One serious problem was the default inittab setting the machine into multiuser state on startup. This can be seen as a security measure, since most small computers will not have a secure room of their own. It can cause problems if, for any one of a number of reasons the system supervisor could not logon as root. The inittab file can be easily changed to allow a single user mode albeit with reduced security.
- 5 There was no online manual available. The manual command 'man' existed but the manual directories are empty. This was true of all Unix systems encountered for these machines. Possibly the size (approximately 2 Megabytes) is considered too large for the small disks of these machines. With secondary disk storage decreasing in cost the inclusion of an online manual becomes more viable. The one copy of the printed manual becomes a valuable item, much sought after by the beginner users.
- 6 No disk usage accounting programmes are provided even though they are documented in the printed manual.
- 7 Some minor programmes would not run and usually dumped core. They are not significant enough to hinder the system's use but are an indication of a lack of refinement of the current command set. They should either be fixed or removed from the next release.

User interaction and education.

This is a key area of the system. It creates problems that extend beyond the purely technical because it involves interaction between users at different levels of computing skills. Some users will have computer skills from their use of MSDOS and CPM. Others will have skills from

mainframe computer use. Neither area will map directly to this small but multiuser environment. A multiuser system usually requires specialist support staff. The small size of this system cannot justify dedicated support staff, so the support will come from the users themselves. A number of users sharing the critical functions associated with the super user account will remove the 'key-man' problem, allowing the system to be maintained as staff come and go. At first there will be more tasks for the super user to perform especially if there is only one such person. Therefore users keen to provide time and effort for administration should be encouraged and trained. The extent that users participate in the traditional role of the super user is yet to be determined. The support will have to come from within and the traditional model for multiuser systems will require change. If the hardware, operating system software and application software remain stable and reliable then the success may depend upon the users attitude and support.

Conclusions.

The domination of Unix on personal computers, as predicted at an AUUGN meeting in 1984, has not occurred. There may have been valid reasons for this when the standard offering was an underpowered, small capacity machine and the few Unix implementations available for these machines cost nearly as much as the machine itself. This situation has now changed. The advent of the AT design, its low cost and the low cost and reasonable range of Unix products available for this design give the designer of a personal or small multiuser Unix system many choices. It is now up to the users to accept that change is possible and make it.

Benchmarking Visual Editors

Albert Nymeyer
University of Newcastle

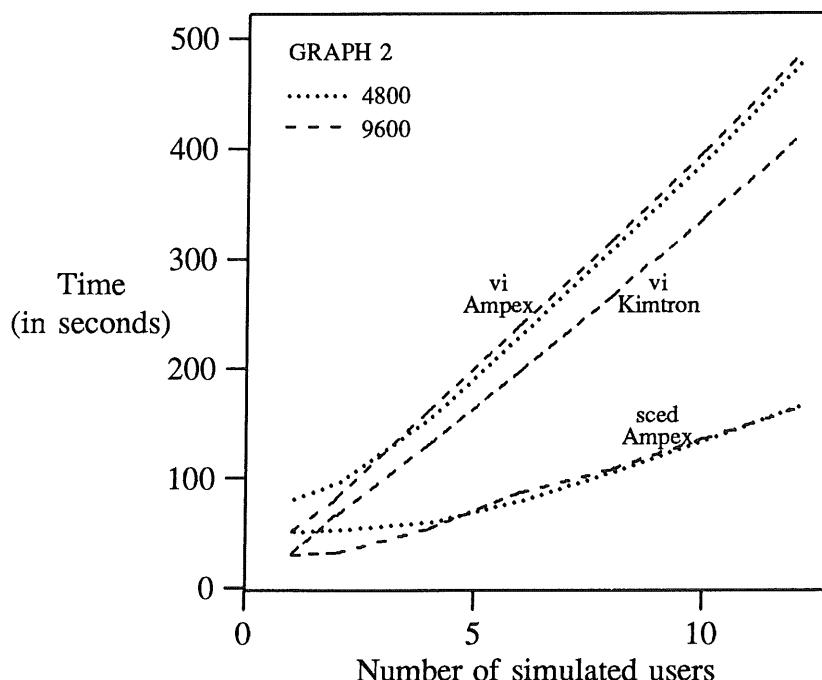
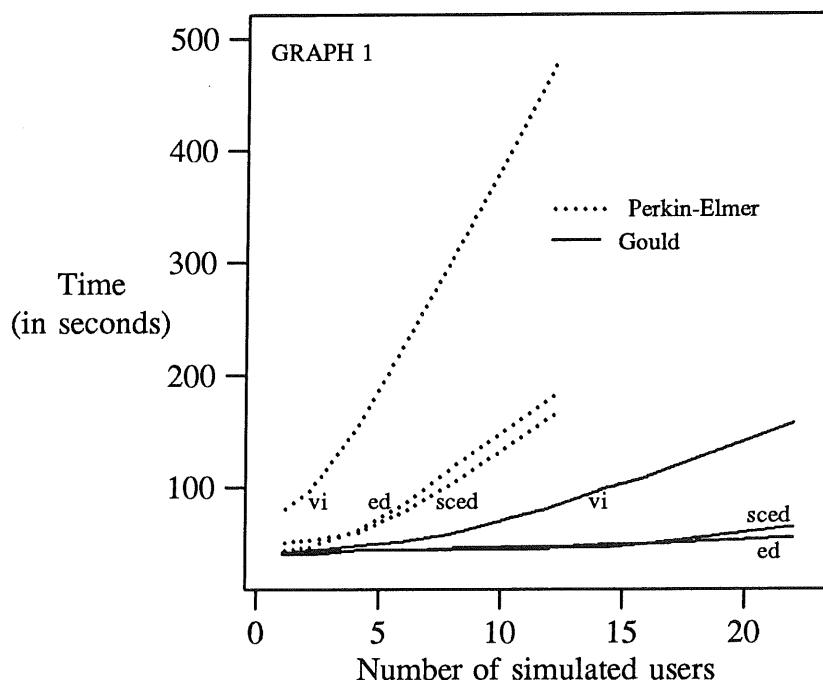
In a typical academic multiuser environment computer users spend most of their logged-in time using a screen editor. During prime time therefore, the performance of the screen editor is a major factor in the overall performance of the system. The standard screen editor at most UNIX sites is *vi*. At our site however, all first year computer science and mathematics students use a more restrictive screen editor called *sced* (SCreen EDitor), originally written by Richard Miller at the University of Wollongong. Like *vi*, *sced* uses the *termcap* database, although not as extensively. *Ed* was also included in this benchmark for comparison purposes. Both *vi* and *sced* were trivially modified to read from command files. This of course means that in this benchmark the CPU time needed to handle interrupts from terminal input will not be a factor. A "typical" sequence of edit instructions were placed in a command file for each of the three editors. Each editor was then run, reading from its own command file, for alternately 1, 2, 4, 6, 8 and so on concurrent users, one per terminal (with no other users logged on). Note that "real" users were used to start the edit scripts on each of the terminals (my thanks to those who gave of their time).

The test was carried out on our main computer, a Gould 9005 (8mb memory), and on a Perkin-Elmer 3220 (1mb memory). The Gould runs 4.2BSD, the Perkin-Elmer runs Level 7. We simulated up to 22 users on the Gould, and up to 12 users on the Perkin-Elmer. The benchmark was run a number of times on the Perkin-Elmer. We tried different baud rates, and different terminals. Two kinds of terminals were used on the Perkin-Elmer, Kimtron KT7 terminals and Ampex Dialogue 30 terminals. Only Kimtron terminals were used on the Gould. The Kimtron terminals are "clever" in the sense that they have insert and delete capabilities, reverse scroll and many other features to speed up screen IO. The Ampex terminals have virtually no features, apart from being cursor addressable.

In designing an edit script for each of the three editors, it is very easy to favour one. *Ed*, for example, being a line editor tends to do very little terminal IO. Scripts of similar content were written for each of the editors. Each was designed to run for 40 seconds real time on the Gould, and each included a liberal number of screen redraws (even *ed*), something that first year students are prone to do. The same edit scripts were used on both machines.

The first graph compares the Perkin-Elmer using Ampex terminals with the Gould using Kimtron terminals all running at 4800 baud. Quite clearly, *vi* is too heavy a load for the Perkin-Elmer (the average job time scales with the number of users), and *sced* is faster than *ed* after about three users. On the Gould *vi* is heavier on the system than *sced*, but not prohibitively so.

In the second graph we show results for the Perkin-Elmer using Kimtron and Ampex terminals running at 4800 and 9600 baud. Increasing the baud rate from 4800 to 9600 improves the performance on a lightly loaded system, but has virtually no effect under heavy loads. The difference between clever and dumb terminals on the Perkin-Elmer is almost constant. On both systems *sced* obviously performs extremely well. The limiting factor in the system is the time it takes the CPU to handle interrupts for screen output. Since every job in a given run has the same output, the resulting curves will be almost linear. It is interesting to note that the addition of two small CPU-intensive jobs running in the background at lowest priority made virtually no difference to the results.



Towards a standard programming interface between Graphics Programs and Graphics Devices.

Steven Bodnar

La Trobe University

I am not proposing a standard that can be compared to GKS or CORE, but sticking to a standard interface when trying to write converters for various graphic devices for various typesetting programs, reduces the total amount of work needed. The standard and the appropriate programs has only been tested on a Pyramid, so portability is questionable.

Here is the proposed data structures and procedure calls:

```
typedef unsigned int word;

struct image {
    int xsize;
    int realxsize;
    int ysize;
    word *data;
};

void
InitDev(xresolution, yresolution, xsize, ysize)
int *xresolution, *yresolution, *xsize, *ysize;

void
FinishDev(flag)
int flag;

void
PrintImage(display)
struct image *display;
```

The 'word' type is a 32 bit quantity. 'data' is a pointer to a set of words that holds a bit image. The bit image size is 'realxsize' by 'ysize' words which holds 'xsize' by 'ysize' bits. Each row should be padded out to a 32 bit boundary. For example, the follwing bit image has a ysize of seven an xsize of five and a realxsize of one.

```
.XXX.
X..X
X..X
.XXXX
....X
X..X
.XXX.
```

Up to date a dvi-format driver, a troff-format driver and a general bit image driver has been written to use this interface. These routines have been written for Philip's gp300 printer, Apple 15inch Imagewriter and the Vextrix Graphics terminal with a more like facility. Future printers will include PostScript printers.

As people write decoders for other typesetters or graphics packages, using this interface will simplify obtaining a translater to a particular printer/terminal. Instead of the famous $N * M$ programs to be written given N formats and M printers, we have $N + M$.

CALL FOR EVALUATORS

I am quite happy to let these programs be freely available for no cost. They can't be sold however. There is a set of programs to resize vfont files to an arbitrary size so do not worry too much if you do not have a 200 dpi printer. At the moment only reduction of vfonts is available.

I would be interested in a full test on a VAX system, as it requires only even byte boundary restrictions instead of pyramid's 4 byte boundary restriction. More importantly there might very well be dependency for the order of the bytes. As the Vax and the Pyramid differ in this respect, it would be a good test. To minimize frustration distribution should be minimal at first to test portability.

A cursory view of the state of Unix on the MV/2000, a Data General Computer

*Steve Bodnar
Philip Lee*

La Trobe University

The MV/2000 is a 24 user system, with a LAN board, winchester drive and cartridge tape. The one feature that strikes you when you first look at it is the non-DB25 RS232 connectors. These are more expensive than conventional DB25 and of course a special crimping tool is needed.

Basically there are two asynchronous serial boards, of 12 ports each. Only two of which have full modem control. The others have only Transmit, Receive and Ground.

On the system board there are four other serial ports, including the Console, which have modem control. It also have one parallel printer port and an SCSI interface port.

The hardware architecture only allows 32 bit addressing, it could be 16 bit but assume 32 for the sake of argument. Rumour has it that early C compilers made characters 32 bits long to circumvent this problem. Now microcode handles 8 bit addressing properly. Unfortunately this means the format for byte addressing is different to the format for word addressing.

```
int    *iptr;
char   *cptr;

iptr = cptr;
```

This will not work, as it will give a completely wrong address.

```
int    *iptr;
char   *cptr;

iptr = (int *) cptr;
```

The above fragment uses inline code to translate the formats and does work, hence ‘lint’ is more important than ever. Of course there is an option to make them all the same format, the microcoded format, rising in poorer performance.

The C compiler is basically taken from there AOS/VS operating system and hammered into place for unix. Consequently there are some options are not available, and many more extra option are available.

Another consequence of taking the machine from an AOS/VS environment to a Unix environment is the Intelligent I/O Hardware. Rumour has it that it is disabled to allow Unix to work properly.

The unix system said to be 100% System 5.2 and X% Berkley 4.2. The irritating part is that you cannot work with 4.2 tools and programs without bumping into something that is missing, but that applies to all partial ports of Berkley Unix.

Unfortunately they mix System V and 4.2 BSD terminal drivers depending whether you are in ‘sh’ or in ‘csh’. Consequently never try change the characteristics of a terminal that only has a getty on it, it has neither driver.

As far as the claim for 100% System V.2, it seems that everything is there. Unfortunately they still have an unreasonable proportion of bugs, but here I must state the we have the pre-release version. TCP/IP is available but does not work with the pre-release version. It cannot be stated with any certainty when the official release will occur.

For use for students we are concentrating on Pascal using Dbx as the debugger. Dbx does not interface with Pascal in the pre-release version, and that problem is reported as being looked into.

Problems with DG terminals can be summed up by saying the ^Y is their backspace and ^Z is generated by there down arrow. Consequently they have special entries in their terminal drivers to allow for such problems.

Unfortunately we cannot show any type of benchmarking because with the pre-release version, we have not yet got the benchmarks to work. In all fairness we spend a lot more time trying to get sunIII to compile and run and hence do very little in the benchmarking department.

CONCLUSION

Our conclusions are that with a more UNIX orientated C compiler and more work on their releases should see the MV/2000 as a Unix computer instead of close but not quite Unix Computer. Again in all fairness we have not been using it much, coupled with the fact that we have the pre-release version of unix means that this is only a current view and possibly an inaccurate view.

Maintaining Geographically Scattered UNIX systems

Ron Baxter & Mark Andrews
CSIRO Division of Mathematics and Statistics, Sydney, NSW

February 21, 1987

Summary

The Division of Mathematics and Statistics (DMS) has a total staff around Australia of nearly 90, and maintains UNIX systems in Canberra, Sydney, Melbourne, Adelaide, Perth, and Brisbane. These systems are used for data analysis, graphics, number crunching, mathematical word-processing and networking. There are about 11 people involved in system-management and user-assistance (about 6 full-time equivalents), but the experience and expertise of these people is varied so that the manpower and expertise at each site varies widely. Consequently, there is a clear need for *distributed management*.

ACSnet is used to interconnect the systems (using CSIRONET as the carrier), and all systems may be accessed from remote locations via CSIRONET and/or dial-up lines. Extensions to ACSnet (originally from UNSW) have been added to allow remote file copy and remote execution of commands. Examples will be presented to show how these facilities are being used to monitor system performance, and to maintain software at all locations.

1 Setting the Scene

The Division of Mathematics and Statistics of CSIRO not only undertakes mathematical research, but also provides a consulting service to research workers in other fields (largely within CSIRO). Since consultants need to be close to their major clients, the Division has staff scattered around Australia. Table 1 shows the locations and computing facilities.

Table 1

Location	Staff		System Details
	Total	System	
Canberra	19	($\frac{1}{2} + \frac{1}{2}$)	Vax 750 (1982) with 4.2 BSD
Sydney	20	(1+ $\frac{1}{2}$)	Vax 750 (1983) with 4.2 BSD
Melbourne	15	(1)	Microvax II (1985) with ULTRIX 1.1
Adelaide	14	($\frac{1}{2} + \frac{1}{2}$)	Microvax II (1986) with UTRIX 1.1
Brisbane	6	($\frac{1}{2}$)	Integrated Solutions 68010 Qbus system (1984) with a port of 4.2 BSD
Perth	6	($\frac{1}{2}$)	Integrated Solutions 68010 Qbus system (1984) with a port of 4.2 BSD
Hobart	3		Access to a local VMS system
Townsville	1		Access to a local VMS system

2 The Problems

Expertise for managing the systems is spread unevenly. Since 6% of the total staff of the Division is involved it would be unrealistic to try and solve this problem by requesting more staff.

While it is obvious that the problems would be reduced if the System details at each site were identical, this is not possible because the staff numbers vary, the specific local needs vary, and the systems are purchased at different times. Nevertheless, we do give high priority to trying to keep as much uniformity as possible.

These problems must be overcome by exploiting network links to allow at least some management tasks to be done remotely.

3 The Tools Available

CSIRONET-UNIX gateways. These gateway connections use one RS232 line from UNIX and one from a CSIRONET micronode, and provide 8 data channels on this single physical line. At the UNIX end we use the *zt* (or *mx*) driver. This allows four UNIX users to access any CSIRONET service, and four CSIRONET users to access the UNIX system. We use these facilities to provide ACSnet links, and we can get throughputs of 30–60 characters per second. With earlier versions of the *zt* driver we could only get 7-bit data paths, but now we have no difficulty with 8-bit transfers.

2400 baud modems. While we can use the CSIRONET gateways to obtain interactive access to remote machines, there are limitations. Using *vi* is near to impossible for example. So we have dialin access on all systems, and use them extensively for

interactive access. They can also be used for ACSnet links in emergencies, but we currently restrict this use to manually initiated connections, because we have had difficulties in the past with phone connections that do not terminate correctly.

ACSnet. The six locations are interconnected using ACSnet. We currently maintain 11 links among these systems (the maximum number is 15 and the minimum number is 5).

Remote Execution and Remote File Copy. These commands *nsh* and *ncp* are provided as extensions to ACSnet and are based on code from UNSW and have syntax as follows:

Usage: *nsh* [-svpS] [-f file] [-i file] :host ... [commands ...]

Usage: *ncp* [-svpS] [:host:]file [:host:][file]
ncp [-svpS] [:host:]file ... [:host:][dir]

The host specification can be a comma separated list, and the **-f** and **-i** options allow the commands and/or the input to be in files.

4 Installing a new version of the News Software

This has been done from *natmlab* where the source code is installed, compiled, and then binaries sent to *dmscanb*. Copies of the source are sent to *dmsmelm* and *dmsbris* where further compilations occur, and then binaries from *dmsmelm* are sent to *dmsadel*, and binaries from *dmsbris* are sent to *dmspertz*.

Apart of the master Makefile is:

```
src: UNITY.src UVAXII.src

make: UNITY.make UVAXII.make

binary: UNITY.binary UVAXII.binary VAX750.binary

UNITY.src: ${SRC1}
ncp -S $? :dmsbris:${REMOTE}
date > $@

UNITY.localize: localize.unity
ncp -S $? :dmsbris:${REMOTE}/localize.sh
-nsh -vS :dmsbris cd ${REMOTE} ";" sh localize.sh
date > $@

UNITY.make: UNITY.src UNITY.localize
-nsh -Sv :dmsbris cd ${REMOTE} ";" make
date > $@
```

```

UNITY.binary: UNITY.localize UNITY.remote
ncp -S UNITY.remote :dmsbris:${REMOTE}/.remote
-nsh -Sv :dmsbris cd ${REMOTE} ";" make -f .remote $@
date > $@

Makefile.binary: Makefile
sed 's/^all:.*/all:/' $? > $@

VAX750.binary: ${BINARIES} ${HELP} ${INSTALL} Makefile.binary
ncp -S $? :dmscanb:${REMOTE}
-nsh -Sv :dmscanb cd ${REMOTE} ";" cp Makefile.binary Makefile
date > $@

UVAXII.src: ${SRC1}
ncp -S $? :dmsmelb:${REMOTE}
date > $@

<lines deleted>

dmsbris.install:
-nsh -S :dmsbris -f INSTALL

dmsmelb.install:
-nsh -S :dmsmelb -f INSTALL

```

5 Monitoring Network Performance

The *linkstats* command of ACSnet was designed to give summary of the health of local network connections. Its default output usually fits onto a single screenful. However, in our context it is not so useful because

- our network consists of intermittent links and *linkstats* only gives information on active links or on the last session of an intermittent link. This may be useless because the most recent session may have done nothing – but this may be correct.
- we need to know more than just the direct links to our current machine. For example, the dmscanb-dmsbris link failed recently — but from where we are in Sydney we did not notice until someone yelled.

In an attempt to generate a more useful summary (still in one screenful), each of the six sites runs a daily cronjob to extract some information relating to direct links from the statefile and *ncp*'s it back to *natmlab*. Awk scripts then try and summarize in a way that will make anomalies easy to observe.

6 Limitations of Remote Gurus

Clearly a remote guru can do nothing more than talk on the phone when the system has crashed, or a file-system is trashed, or there appears to be a hardware failure.

Recently we were told that following a thunderstorm in Brisbane, the response time of *dmsbris* was extremely slow. This was easy to confirm by trying to login. Eventually we discovered the cause was a *runaway* getty caused by characters arriving on one of the serial lines. The way to identify and fix this problem was to login as root and renice to -20 — this took ages, but once the renice had happened, the problem was close to solved.

Preface to C++ Overview Paper

Permission to re-publish the following paper was kindly given by Bjarne Stroustrup of AT&T Bell Laboratories, Murray Hill, New Jersey 07974. It appeared originally in SigPlan Notices, Oct 1986. I have altered one footnote and added a couple of others. These may be distinguished by the "MM" that appears in them.

The paper is notable for the way it explains the motivation for C++ starting from the reasons why so many people use C. Indeed, I believe that anyone even vaguely familiar with C would thoroughly enjoy programming in C++, from the point of view of satisfaction (expressiveness and efficiency), *as well as* from productivity and maintainability (and hence economic) perspectives.

The separate treatment of C++'s support for data abstraction and for object-oriented programming is also instructive. (Far too often, these two terms are used interchangeably by people who should know better.) C++ does not provide *methods* which are common to most other "object-oriented" languages, but instead uses a *virtual function* mechanism which is almost as efficient as a conventional function call. Personally, I regard the *method* approach as slow and potentially dangerous, being un-checkable at compile-time. C++ does not suffer this, and manages to provide a truly object-oriented capability for dealing with instances of objects in an inheritance hierarchy where the exact types of the instances are unknown at compile-time.

The current user-base is quite large (over 1500 installations outside AT&T at least) given that C++ has only been commercially available for little more than a year or so. C++ is available from AT&T Unix Pacific (address given in a footnote in the paper). My most recent pricing information for source code (unsupported) is \$US 2000 per CPU for commercial licenses, and \$US 250 for educational institutions (any number of CPU's). They also offer modestly-priced binary licenses for AT&T machines. Olivetti don't sell it, which I find inexplicable.

\$US 2000 is probably not an unreasonable price for the source code. Consider one programmer costing his/her employer only \$30K pa, say. C++ has only to give that person a 10% productivity gain over a year to have paid for itself, everything else being equal. Since C++ is link-compatible with C, there is no need to rewrite existing code. My own experience¹ indicates that C++ would give much more than a mere 10% gain, even including the learning curve. I have also found that the nature of C++ lessens the amount of reliance on symbolic debuggers. (Such utilities can still be used for C++, of course, although this is not for the novice. Better compile-time checking, plus the better program structure that C++ engenders, more than compensate for the extra difficulty in using existing C debuggers.) Also, there would, of course, normally be more than just one programmer benefiting (probably on higher salaries), and over much longer than one year. Refraining from using C++ because of cost is definitely false economy.²

-
1. Look for an article on the use of C++ applied to implementing layered protocols coupled with non-trivial network management structures in a future issue of AUUGN.
 2. There was a list of humorous language descriptions on the net some time ago, along the lines of: Fortran - the original model-T Ford, still chugging along; Basic - the beat-up VW your dad bought you, you'll buy something better when you can afford it; Forth - a go-cart, easy to use; C - a black Firebird, the all-macho car, comes with optional seat belts (lint); etc. Given these predecessors, one might describe C++: a black and silver Firebird, the all-macho car with class. Comes with inertia-reel seat belts, and you can add more cylinders, or enlarge the boot, or increase the internal cabin size Tardis-fashion, with little more than a screw-driver.

I take this opportunity to remind conservative programmers of a quotation used by Bjarne in his book on C++ (referenced at the end of the paper):

Language shapes the way we think, and
determines what we can think about.

- *B.L.Whorf*

Until one has explored the efficient, expressive, object-oriented features of C++, one is virtually stumbling around in a dungeon, believing it to be the best environment in this imperfect world, and claiming that striking a match occasionally is sufficient to see all that can be seen. Such a person simply cannot conceive of wide-open spaces or large structures at all, and therefore cannot imagine cities, etc. (Not that C++ claims to be perfection, but it *is* a considerable advance, which anyone engaged in serious software engineering should examine very closely.)

Mike Mowbray
Senior Engineer
Systems Development
Overseas Telecommunications Commission (Australia)
Box 7000 GPO Sydney.

Tel: 02-230-4104

ACSnet: mikem@otc.oz

(For people who receive the news on ACSnet, there is a newsgroup *comp.lang.c++*)

Disclaimer: The opinions expressed in this preface are not necessarily those of my employer OTC.

An Overview of C++

Bjarne Stroustrup

AT&T Bell Laboratories

1. Introduction

C++ is a general purpose programming language^[1] designed to make programming more enjoyable for the serious programmer. Except for minor details, C++ is a superset of the C language^[2]. C++ was designed to

1. be a better C.
2. support data abstraction.
3. support object-oriented programming.

This paper describes the features added to C to achieve this. In addition to C, the main influences on the design of C++ were Simula67^[3] and Algol68.^[4]

C++ has been in use for about four years and has been applied to most branches of system programming including compiler construction, data base management, graphics, image processing, music synthesis, networking, numerical software, programming environments, robotics, simulation, and switching. It has a highly portable implementation and there are now at least 1500 installations including AT&T 3B, DEC VAX, Pyramid, Sun, Intel 80286, Motorola 68000, and Amdahl machines running UNIX¹ and other operating systems.²

2. What is Good about C?

C is clearly not the cleanest language ever designed, nor the easiest to use, so why do so many people use it?

1. C is flexible: It is possible to apply C to most every application area, and to use most every programming technique with C. The language has no inherent limitations that preclude particular kinds of programs from being written.
2. C is efficient: The semantics of C are "low level"; that is, the fundamental concepts of C mirror the fundamental concepts of a traditional computer. Consequently, it is relatively easy for a compiler and/or programmer to efficiently utilize hardware resources for a C program.
3. C is available: Given a computer, whether the tiniest micro or the largest super-computer, the chance is that an acceptable quality C compiler is available and that that C compiler supports an acceptably complete and standard C language and library. There are also libraries and

1. UNIX is a Trademark of AT&T Bell Laboratories.

2. For Australia, C++ is available from AT&T Unix Pacific, 2-21-2 Nishi-Shinbashi, Minato-Ku, Tokyo 105 Japan.
Tel: 03-431-3305; Fax: 03-431-3680; Tlx: J34936 ATTUP;
ACSN: upshowal!schwarz@munnari

The C++ Translator generates ordinary C which can then be compiled using a normal C compiler supporting flexnames and structure assignment/passing. (This process is managed by a script, and is of no inconvenience to the user, who sees just an extended form of the normal cc features. Note that this two-stage compilation process does *not* compromise efficiency in itself.) If you have such a c-compiler you should be able to get the C++ source code compiled and running within a few days at the most. I understand that Oasys have done ports to VMS, MS-DOS, Macintosh, etc, but I haven't yet used these, nor spoken with anyone who has.) At the time of writing this footnote, (31st Jan 1987) the current release is 1.1, but I suspect that by the time this appears in print, 1.2 may be available. - MM

support tools available, so that the programmer rarely needs to design a new system from scratch.

4. C is portable: A C program is not automatically portable from one machine (and operating system) to another, nor is such a port necessarily easy to do. It is, however, usually possible, and the level of difficulty is such that porting even major pieces of software with inherent machine dependencies is typically technically and economically feasible.

Compared with these "first-order" advantages, any "second-order" drawbacks like the curious C declarator syntax and the lack of safety of some language constructs become less important. Designing "a better C" implies compensating for the major problems involved in writing, debugging, and maintaining C programs *without compromising the advantages of C*. C++ preserves all these advantages and compatibility with C at the cost of abandoning claims to perfection and of some compiler and language complexity. However, designing a language "from scratch" does not ensure perfection and the C++ compilers compare favorably in run-time, have better error detection and reporting, and equal the C compilers in code quality.

3. A Better C

The first aim of C++ is to be "a better C" by providing better support for the styles of programming for which C is most commonly used. This primarily involves providing features that make the most common errors unlikely. (Since C++ is a superset of C such errors cannot simply be made impossible.)

Argument Type Checking and Coercion

The most common error in C programs is a mismatch between the type of a function argument and the type of the argument expected by the called function. For example:

```
double sqrt(a) double a;
{
    /* .... */
}

double sq2 = sqrt(2);
```

Since C does not check the type of the argument 2 the call `sqrt(2)` will typically cause a run-time error or give a wrong result when the square root function tries to use the integer 2 as a double precision floating point number. In C++, this program will cause no problem since 2 will be converted to a floating point number at the point of the call. That is, `sqrt(2)` is equivalent to `sqrt((double)2)`.

Where an argument type does not match the argument type specified in the function declaration and no type conversion is defined the compiler issues an error message. For example, in C++, `sqrt("Hello")` causes a compile-time error.

Naturally, the C++ syntax also allows the type of arguments to be specified in function declarations:

```
double sqrt(double);
```

and a matching function definition syntax is also introduced:

```
double sqrt(double d)
{
    // ....
}
```

Inline Functions

Most C programmers rely on macros to avoid function call overhead for small frequently-used operations. Unfortunately the semantics of macros are very different from the semantics of

functions so the use of macros has many pitfalls. For example:

```
#define mul(a,b)  a*b  
int z = mul(x*3+2, y/4);
```

Here `z` will be wrong since the macro will expand to `x*3+2*y/4`. Furthermore, C macro definitions do not follow the syntactic rules of C declarations, nor do macro names follow the usual scope rules. C++ circumvents such problems by allowing the programmer to declare `inline` functions:

```
inline int mul(int a, int b) { return a*b; }
```

An inline function has the same semantics as a "normal" function but the compiler can typically inline-expand it so that the code-space and run-time efficiency of macros are achieved.

Scoped and Typed Constants

Since C does not have a concept of a symbolic constant, macros are used. For example:

```
#define TBLMAX (TBLSIZE-1)
```

Such "constant macros" are neither scoped nor typed and can (if not properly parenthesized) cause problems similar to those of other macros. Furthermore, they must be evaluated each time they are used and their names are "lost" in the macro expansion phase of the compilation and consequently are not known to symbolic debuggers and other tools. In C++ constants of any type can be declared:

```
const int TBLMAX = TBLSIZE-1;
```

Varying Numbers of Arguments

Functions taking varying numbers of arguments and functions accepting arguments of different types are common in C. They are a notable source of both convenience and errors.

C functions where the type of arguments or the number of arguments (but not both) can vary, can be handled in a simple and type-secure manner in C++. For example, a function taking one, two, or three arguments of known type can be handled by supplying default argument values which the compiler uses when the programmer leaves arguments out. For example:

```
void print(char*, char* = "-", char* = "-");  
  
print("one","two","three");  
print("one","two");           // I.e: print("one","two","-");  
print("one");                // I.e: print("one","-","-");
```

Some C functions take arguments of varying types to provide a common name for functions performing similar operations on objects of different types. This can be handled in C++ by *overloading* a function name. That is, the same name can be used for two functions provided the argument types are sufficiently different to enable the compiler to "pick the right one" for each call. For example:

```
overload print;  
void print(int);  
void print(char*);  
  
print(1);    // integer print function  
print("two"); // string print function
```

The most general examples of C functions with varying arguments cannot be handled in a type-secure manner. Consider the standard output function `printf`, which takes a format string followed by an arbitrary collection of arguments supposedly matching the format string³:

```

printf("a string");
printf("x = %d\n", x);
printf("name: %s\n size: %d\n", obj.name, obj.size);

```

However, in C++ one can specify the type of initial arguments and leave the number and type of the remaining arguments unspecified. For example, `printf` and its variants can be declared like this:

```

int printf(const char* ...);
int fprintf(FILE*, const char* ...);
int sprintf(char*, const char* ...);

```

These declarations allow the compiler to catch errors such as

```

printf(stderr, "x = %d\n", x); // error: printf does not take a FILE*
fprintf("x = %d\n", x); // error: fprintf needs a FILE*

```

Declarations as Statements

Uninitialized variables are another common source of errors. One cause of this class of errors is the requirement of the C syntax that declarations can only occur at the beginning of a block (before the first statement). In C++, a declaration is considered a kind of statement and can consequently be placed anywhere. It is often convenient to place the declaration where it is first needed so that it can be initialized immediately. For example:

```

void some_function(char* p)
{
    if (p==0)
        error("p==0 in some_function");
    int length = strlen(p);
    // .....
}

```

4. Support for Data Abstraction

C++ provides support for data abstraction: the programmer can define types that can be used as conveniently as built-in types and in a similar manner. Arithmetic types such as rational and complex numbers are common examples:

```

class complex {
    double re, im;
public:
    complex(double r, double i) { re=r; im=i; }
    complex(double r) { re=r; im=0; } // float->complex conversion.

    friend complex operator+(complex,complex);
    friend complex operator-(complex,complex); // binary minus
    friend complex operator-(complex); // unary minus
    friend complex operator*(complex,complex);
    friend complex operator/(complex,complex);

};

```

^{3.} A C++ I/O system that avoids the type insecurity of the `printf` approach is described in reference [1]

The declaration of class (that is, user-defined type) `complex` specifies the representation of a complex number together with the set of operations *on* a complex number. The representation is *private*; that is, `re` and `im` are accessible only to the functions defined in the declaration of class `complex`. Such functions can be defined like this:

```
complex operator+(complex a1, complex a2)
{
    return complex(a1.re+a2.re, a1.im+a2.im);
}
```

and used like this:

```
main()
{
    complex a = 2.3;
    complex b = 1/a;
    complex c = a+b+complex(1,2.3);
    // .....
}
```

Functions declared in a class declaration using the keyword `friend` are called *friend functions*. They do not differ from ordinary functions except that they may use private members of classes that name them friends. A function can be declared as a friend of more than one class. Other functions declared in a class declaration are called *member functions*. A member function is in the scope of the class and must be invoked for a specific object of that class.

Initialization and Cleanup

When the representation of a type is hidden, some mechanism must be provided for a user to initialize variables of that type. A simple solution is to require a user to call some function to initialize a variable before using it. This is error-prone and inelegant. A better solution is to allow the designer of a type to provide a distinguished function to do the initialization. Given such a function, allocation and initialization of a variable become a single operation (often called instantiation) instead of two separate operations. Such an initialization function is called a constructor. Also, in cases where construction of objects of a type is non-trivial one often needs a complementary operation to clean up objects after their final use. In C++ such a cleanup function is called a destructor. Consider a vector type:

```
class Vector {
    int sz;           // number of elements
    int *v;           // pointer to integers
public:
    Vector(int);    // constructor
    ~Vector();       // destructor
    // .....
};
```

The Vector constructor can be defined to allocate a suitable amount of space like this:

```
Vector::Vector(int s)
{
    if (s<=0)
        error("bad Vector size");
    sz = s;
    v = new int[s];    // allocate an array of "s" integers
}
```

The cleanup done by the Vector destructor consists of freeing the storage used to store the vector elements, for re-use by the free-store manager:

```

Vector::~Vector()
{
    delete v;
}

```

Clearly, C++ does not support garbage collection. This is, however, compensated for by enabling a type to maintain its own storage management without requiring intervention from a user.⁴

Free Store Operators

The operators `new` and `delete` were introduced to provide a standard notation for free store allocation and deallocation. A user can provide alternatives to their default implementations by defining functions called `operator new` and `operator delete`. For built-in types the `new` and `delete` operators provide only a notational convenience (compared with the standard C functions `malloc()` and `free()`). For user-defined types such as `Vector`, the free store operators ensure that constructors and destructors are called properly:

```

Vector* fct1(int n)
{
    Vector v(n);                      // Allocate a vector on the stack.
                                         // The constructor is called.

    Vector* p = new Vector(n);        // Allocate a vector on the free store.
                                         // The constructor is called.
    // .....

    return p;                         // The destructor is implicitly called for "v" here.
}

Vector* fct2()
{
    Vector* pv = fct(10);
    // .....
    delete pv; // call the destructor and free the store.
}

```

References

C provides (only) "call by value" semantics for function argument passing; "call by reference" can be simulated by use of pointers. This is sufficient, and often preferable to using "pass by value" for the built-in types of C. However, it can be inconvenient for larger objects⁵ and can get seriously in the way of defining conventional notation for user-defined types in C++. Consequently, the concept of a "reference" is introduced. A reference acts as a name for an object; `T&` means reference to `T`. A reference must be initialized and becomes an alternative name for the object it is initialized with. For example:

```

int a = 1; // "a" is an integer initialized to 1
int& r = a; // "r" is a reference initialized to "a"

```

The reference `r` and the integer `a` can now be used in the same way and with the same meaning. For example:

⁴. The details of this are beyond the scope of an overview paper, however. - MM

⁵. as indicated by an inconsistency in the C semantics: arrays are always passed by reference.

```

int b = r;           // "b" is initialized to the value of "r", i.e: 1
r = 2;              // The value of "r", i.e: the value of "a" becomes 2

```

References enable variables of types with "large representations" to be manipulated efficiently without explicit use of pointers. Constant references are particularly useful:

```

Matrix operator+(const Matrix& m1, const Matrix& m2)
{
    // code here cannot modify the value of "m1" or "m2"
}

Matrix a = b+c;

```

In such cases the "call by value" semantics are preserved while achieving the efficiency of "call by reference".

Assignment and Initialization

Controlling construction and destruction of objects is sufficient for many, but not all, types. It can also be necessary to control all copy operations. Consider:

```

Vector v1[100]; // make v1 a vector of 100 elements
Vector v2 = v1; // make v2 a copy of v1;
v1 = v2;         // assign v1 to v2 (i.e: copy the elements)

```

Declaring a function with the name `operator=` in the declaration of class `Vector` specifies that vector assignment is to be implemented by that function:

```

class Vector {
    int sz;
    int *v;
public:
    // .....
    void operator=(Vector&); // assignment
};

```

Assignment might be defined like this:

```

void Vector::operator=(Vector& a) // check size and copy elements
{
    if (sz != a.sz)
        error("Bad Vector size for =");
    for (int i=0; i<sz; i++)
        v[i] = a.v[i];
}

```

Since the assignment operation relies on the "old value" of the vector assigned to, it cannot be used to implement initialization of one vector with another. What is needed is a constructor that takes a vector argument:

```

class Vector {
    // .....
    Vector(int);           // create vector
    Vector(Vector&);     // create vector and copy contents
};

Vector::Vector(Vector& a) // initialize a vector from another vector
{
    sz = a.sz;            // same size.
    v = new int[sz];       // allocate element array.
    for (int i=0; i<sz; i++)
        v[i] = a.v[i];     // same values.
}

```

A constructor like this (of the form `x(x&)`) is used to handle all initialization. This includes arguments passed "by value" and function return values:

```

Vector v2 = v1;           // uses Vector(Vector&) constructor to initialize

void f(Vector);
f(v2);                  // uses Vector(Vector&) constructor to pass
                        // a copy of v2

Vector g(int sz)
{
    Vector v(sz);
    return v;           // uses Vector(Vector&) constructor to return
                        // a copy of v.
}

```

Operator Overloading

As demonstrated above, standard operations like `+`, `-`, `*`, `/` can be defined for user-defined types, as can assignment and initialization in its various guises. In general, all the standard operators with the exception of

`->` `.` `,` `?:`

can be overloaded. The subscripting operator `[]` and the function application operator `()` have proven particularly useful. The C "operator assignment" operators, such as `+=` and `*=`, have also found many uses.

It is not possible to redefine an operator when applied to built-in data types, nor to define new operators, nor to re-define the precedence of operators.

Coercions

User-defined coercions, like the one from floating point numbers to complex numbers implied by the constructor `complex(double)`, have proven unexpectedly useful in C++. Such coercions can be applied explicitly or the programmer can rely on the compiler adding them implicitly where necessary and unambiguous:

```

complex a = complex(1);
complex b = 1;           // implicit: 1 -> complex(1)
a = b+complex(2);
a = b+2;                 // implicit: 2 -> complex(2)
a = 2+b;                 // implicit: 2 -> complex(2)

```

Coercions were introduced into C++ because mixed-mode arithmetic is the norm in languages used for numerical work and because user-defined types used for "calculation" (for example: matrices, character strings, and machine addresses) have natural mappings to and/or from other types.

Great care is taken (by the compiler) to apply user-defined conversions only where a unique conversion exists. Ambiguities caused by conversions are compile-time errors.

It is also possible to define a conversion to a type without modifying the declaration of that type. For example:

```
class Point {
    float dist;
    float angle;
public:
    // .....
    operator complex()           // convert point to complex number
    {
        return polar(dist,angle);
    }

    operator double()           // convert point to real number
    {
        if (angle)
            error("Can't convert Point to real: angle!=0");
        return dist;
    }
};
```

These conversions would be used like this:

```
void some_function(Point a)
{
    complex z = a;           // z = a.operator complex()
    double d = a;            // d = a.operator double()
    complex z3 = a+3;        // z3 = a.operator complex() + complex(3);
    // .....
}
```

This is particularly useful for defining conversion to built-in types since there is no declaration for a built-in type for the programmer to modify. It is also essential for defining conversions to "standard" types where a change may have (unintentionally) wide ranging ramifications and where the average programmer has no access to modify the declaration.

5. Support for Object-Oriented Programming

C++ provides support for object-oriented programming: the programmer can define class hierarchies and a call of a member function can depend on the actual type of an object (even where the actual type is unknown at compile time). That is, the mechanism that handles member function calls handles the case where it is known at compile-time that an object belongs to *some* class in a hierarchy, but exactly *which* class can only be determined at run-time. See examples below.

Derived Classes

C++ provides a mechanism for expressing commonality among different types by explicitly defining a class to be part of another. This allows re-use of classes without modification of existing classes and without replication of code. For example, given a class `Vector`:

```

class Vector {
    // ....
public:
    // ....
    Vector(int);
    int& operator[](int); // overloading the subscript operator: []
};

```

one might define a different vector type for which the user can define the index bounds:

```

class Vec : public Vector {
    int low,high;
public:
    Vec(int,int);
    int& operator[](int);
};

```

Defining `Vec` as

```
: public Vector
```

means that first of all a `Vec` is a `Vector`. That is, type `Vec` has ("inherits") all the properties of type `Vector` in addition to the ones declared specifically for it. Class `Vector` is said to be the *base class* for `Vec` and conversely, `Vec` is said to *derived* from `Vector`.

Class `Vec` modifies class `Vector` by providing a different constructor, requiring the user to specify the two index bounds rather than the size, and by providing its own access function `operator[]()`. `Vec`'s `operator[]()` is easily expressed in terms of `Vector`'s `operator[]()`:

```

int& Vec::operator[](int i)
{
    return Vector::operator[](i-low);
}

```

The scope resolution operator `::` is used to avoid getting caught in a infinite recursion by calling `Vec::operator[]()` from itself. Note that `Vec::operator[]()` *had* to use a function like `Vector::operator[]()` to access elements. It could not just use `Vector`'s members `v` and `sz` directly since they were declared *private* and therefore accessible only to `Vector`'s member functions.

The constructor for `Vec` can be written like this:

```

Vec::Vec(int lb, int hb) : (hb-lb+1)
{
    if (hb-lb < 0)
        hb = lb;
    low = lb;
    high = hb;
}

```

The construct `: (hb-lb+1)` is used to specify the argument list needed for the base class constructor for `Vector()`.

Class `Vec` can be used like this:

```

void some_function(int l, int h)
{
    Vec      v1(l,h);
    const int sz = h-l+1;
    Vector   v2(sz);
    // .....
    for (int i=0; i<sz; i++)
        v2[i] = v1[l+i]; // copy elements explicitly

    v2 = v1;           // copy elements by using Vector::operator=()
}

```

Virtual Functions

Class derivation (often called subclassing) is a powerful tool in its own right but a facility for run-time type resolution is needed to support object-oriented programming.

Consider defining a type `Shape` for use in a graphics system. The system has to support circles, triangles, squares, and many other shapes. First specify a class that defines the general properties of all shapes:

```

class Shape {
    Point  centre;
    Colour col;
    // .....
public:
    virtual void draw();
    virtual void rotate(int);

    Point      where()          { return centre;      }
    void       move(Point to)   { center=to; draw(); }
    // .....
};

```

The functions for which the calling interface can be defined, but where the implementation cannot be defined except for a specific shape, have been marked *virtual* (the Simula67 and C++ term for "to be defined later in a class derived from this one"). Given the definition one can write general functions manipulating shapes:

```

void rotate_all(Shape* v, int size, int angle)
// rotate all members of vector "v" of size "size", "angle" degrees
{
    for (int i=0; i<size; i++)
        v[i].rotate(angle);
}

```

For each `Shape v[i]`, the proper `rotate` function for the actual type of the object will be called. The "actual type" is not known at compile time.

To define a particular shape we must say that it is a `Shape` (that is, derive it from class `Shape` and specify its particular properties (including the virtual functions):

```

class Circle : public Shape {
    int radius;
public:
    void draw() { /* .... */ }
    void rotate(int) {} // yes, the null function
};

```

In many contexts it is important that the C++ virtual function mechanism be very nearly as efficient as a "normal" function call. The additional run-time overhead is about 4 memory references (dependent on the machine architecture and the compiler) and the memory overhead is one word per object instance, plus one word per virtual function per class.

Visibility Control

The basic scheme for separating the (public) user interface from the (private) implementation details has worked out very well for data abstraction uses of C++. It matches the idea that a type is a black box. It has proven to be less than ideal for object-oriented uses.

The problem is that a class defined as part of a class hierarchy is not simply a black box. It is often primarily a building-block for the design of other classes. In this case the simple binary choice *public/private* can be constraining. A third alternative is needed: a member should be private as far as functions outside the class hierarchy are concerned, but accessible to member functions of a derived class in the same way that it is accessible to members of its own class. Such a member is said to be *protected*.

For example, consider a class `Node` for some kind of tree:

```

class Node {
    // private stuff
protected:
    Node* left;
    Node* right;
    // more protected stuff
public:
    virtual void print();
    // more public stuff
};

```

The pointers `left` and `right` are inaccessible to the general user but any member function of a class derived from class `Node` can manipulate the tree without overhead or inconvenience.

The protection/hiding mechanism applies to names independently of whether a name refers to a function or a data member. This implies that one can have `private` and `protected` function members. Usually it is a good policy to keep data `private` and present the `public` and `protected` interfaces as sets of functions. This policy minimizes the effect of changes to a class on its users and consequently maximises its implementor's freedom to make changes.

Another refinement of the basic inheritance scheme is that it is possible to inherit public members of a base class in such a way that they do not become public members of the derived class. This can be used to provide restricted interfaces to standard classes. For example:

```

class DeQueue {
    // ...
    void insert(Elem*);
    void append(Elem*);
    ELEM* remove();
};

```

Given a `DeQueue` a `Stack` can be defined as a derived class where only the `insert()` and `remove()` operations are defined:

```

class Stack : DeQueue { // note: just ":" not ":", public"
    // members of DeQueue are private members of Stack
public:
    DeQueue::insert;    // make "insert" a public member of Stack
    DeQueue::remove;   // make "remove" a public member of Stack
};

```

Alternatively, inline functions can be defined to give these operations the conventional names:

```

class Stack : DeQueue {
public:
    void push(Elem* ee) { DeQueue::insert(ee); }
    Elem* pop()        { return DeQueue::remove(); }
};

```

6. What is Missing?

C++ was designed under the severe constraints of compatibility, internal consistency, and efficiency: no feature was included that

1. would cause a serious incompatibility with C at the source or linker levels.
2. would cause run-time or space overheads for a program that did not use it.
3. would increase run-time or space requirements for a C program.
4. would significantly increase the compile time compared to C.
5. could only be implemented by making requirements of the programming environment (linker, loader, etc) that could not be simply and efficiently implemented in a traditional C programming environment.

Features that might have been provided but weren't because of these criteria include garbage collection, parametrized classes, exceptions, multiple inheritance, support for concurrency, and integration of the language with a programming environment. Not all of these possible extensions would actually be appropriate for C++ and unless great constraint is exercised when selecting and designing features for a language a large, unwieldy and inefficient mess will result. The severe constraints on the design of C++ have probably been beneficial and will continue to guide the evolution of C++.⁶

7. Conclusion

C++ has succeeded in providing greatly improved support for traditional C-style programming without added overhead. In addition, C++ provides sufficient language support for data abstraction and object-oriented programming in demanding (both in terms of machine utilization and application complexity) real-life applications. C++ continues to evolve to meet demands of new application areas. There still appears to be ample scope for improvement even given the (self-imposed) Draconian criteria for compatibility, consistency and efficiency. However, currently the most active areas of development are not the language itself but libraries and support tools in the programming environment.

6. This should not necessarily be taken to mean that none of the features mentioned will ever be added to C++. Some of them (e.g. multiple inheritance) seem highly desirable, but quite difficult to design and implement in such a way that the named criteria are adhered to. The list of features here might also be read as a "wish-list" of future enhancements. - MM

8. Acknowledgements

C++ could never have matured without the constant help and constructive criticism of my colleagues and users; notably Tom Cargill, Jim Coplien, Stu Feldman, Sandy Fraser, Steve Johnson, Brian Kernighan, Bart Locanthi, Doug McIlroy, Dennis Ritchie, Ravi Sethi, and Jon Shopiro. Brian Kernighan and Andy Koenig made many helpful comments on drafts of this paper.

REFERENCES

1. Stroustrup, Bjarne
The C++ Programming Language
Addison-Wesley, 1986
2. Kernighan, B.W. and Ritchie, D.M.
The C Programming Language
Prentice-Hall, 1978
3. Birtwistle, Graham et.al.
SIMULA BEGIN
Studentlitteratur, Lund, Sweden. 1973.
4. Woodward, P.M. and Bond, S.G.
Algol 68-R Users Guide
Her Majesty's Stationery Office, London. 1974.

Document production in the Unix environment

Stephen Frede
Softway Pty Ltd
P.O. Box 305, Strawberry Hills, NSW 2012

1. Introduction

This paper covers a range of areas, all related to document production of one sort or another. Hopefully, at least some parts will be interesting or even useful.

I believe the information contained herein to be accurate, and up to date as of January, 1987. However, of necessity the information is from a wide variety of sources, some of which I have no means of checking, and I take no responsibility for any errors that may exist.

2. Formatters

The most commonly used document formatters under Unix are troff and Te χ (pronounced "tech"). Also available is Scribe, which is available from Unilogic for large sums of money. Scribe is similar in flavour to Te χ , and has been simulated with a Te χ macro package. While some people would use nothing else, most find that the functionality does not justify the cost. Both troff and Te χ have their devotees.

2.1 Troff

The history of troff is a long one. It started out with "roff", written in PDP-11 assembler by Joe Ossanna in Bell Labs. This was used originally for preparing patent applications, and indeed the first PDP11/45 to run Unix, was bought jointly by the computing research and patent applications departments; but that's another story.

When a CAT mechanical phototypesetter was purchased, the first troff was developed to produce output for it. The new program had a somewhat extended and more flexible command set than the old. At the same time, nroff was written to allow previewing of documents, before sending them to the typesetter. Because there were a number of different types of printers available, nroff read in a device table at run-time to determine character codes and widths, and certain other device-specific information. Both these programs were also written in PDP-11 assembler. A macro package was developed to allow backward compatibility with roff. All three programs, roff, troff and nroff, were distributed in Version 6 of Unix.

Then came the VAX, and a great effort was made to translate the assembler code of troff and nroff into C, so that it could be released with Version 7 Unix and Unix 32V. Additionally, the tbl and eqn preprocessors were developed (by Mike Lesk and Lorinda Cherry respectively). It was around this time that Ossanna died in a car accident, and much of the knowledge of the internals of troff was lost.

For several years the situation stayed pretty much the same, with no-one game enough to make any changes to the quite formidable code. However not everyone had a CAT typesetter, and so several postprocessors were developed to convert the troff output into a form suitable for other phototypesetters (eg vgrind for versatecs). However, this was not entirely satisfactory.

The next major development occurred in 1979 with the modification of troff (mainly by Brian Kernighan) to be device independant. This was achieved by reading device specific tables, and generating a generic output language, which is then interpreted by a device specific driver program. At the same time, many limits were relaxed and some new commands added, including some simple graphics primitives. Also, at times new preprocessors and macro packages were added to the troff menagerie, by various people and organisations. The list of troff preprocessors now includes: *tbl*, *eqn*, *refer*, *pic*, *ideal*, *grap*, *bib*, *chem*, *lbl* and *cw*. The new troff (often known as "ditroff", while the old CAT troff is called "otroff") together with the *pic*, *eqn* and *tbl* preprocessors and some macro packages were bundled by AT&T into the Documenter's Workbench (or DWB) and sold separately. It is interesting to note that on Xenix, the dictionary and the "spell" and "cut" commands are also bundled in with troff and friends to form the text processing system. I bet you thought "cut" was a standard Unix tool!

More recently, DWB 2.0 has been released, which additionally includes the "grap" preprocessor, some performance improvements to troff (speed-up factor of about 2), and some new device drivers.

2.2 Te χ

The history of Te χ is very different. Donald Knuth was unhappy with the typesetting of his famous books, and so started investigating the subject of typesetting and document preparation in great detail, to the level of font design. The primary result of these labours is Te χ , a product of considered and careful design, rather than the somewhat quirky development of troff. Friends of Te χ include the macro package LaTe χ , the font description language MetaFont (in which a rune font was recently posted over Usenet), and the programs Web, Weave and Tangle, which I believe are to do with integrating documentation and programs.

Both troff and Te χ have their staunch adherents. Nobody is really happy with troff, but it seems that many people are not willing to invest the time and effort in learning all the intricacies of Te χ , especially when they may just be swapping one set of arcane magic for another.

2.3 Comparison

Here is a brief comparison of some features of DWB and Te χ . Much of this comes from the recent debate on Usenet over the issue.

2.3.1 Ease of Use

Neither troff nor Te χ are simple enough that you can produce non-trivial documents without careful reading of the manuals. In fact most people seem to think that for very simple documents, it is easier to use troff. When it comes to more involved document formatting, they are probably about even in terms of complexity of use, although Te χ macros are easier to read (if not to understand).

2.3.2 Documentation

The documentation for troff is actually fairly good, although this was certainly not always the case. Many introductory books on Unix include a chapter on troff, and some books have been written exclusively on Troff.

Several books have been written on Te χ , and there is a Te χ Users Group (TUG) which prints a regular journal, known as "Tugboat". For a novice user, probably the usual method of solving a problem in either case is the traditional "seek out a guru" approach.

2.3.3 Availability

As far as I know, DWB is only widely available on Unix systems, though there is nothing particularly Unix specific about it, and indeed I seem to remember hearing about a port to MS/DOS. In Australia, DWB 2.0, with a variety of drivers is available from Softway Pty Ltd.

There are a variety of Te χ systems available for various environments, including Unix, MS/DOS, Macintosh, etc. The fact that it was originally written in pascal was an initial hindrance to its acceptance, but C versions are now available, and it is becoming ever more widespread.

2.3.4 Drivers

To run either Te χ or troff with a given printer, you must have available a driver and font tables for that device. The availability of drivers for both formatters is now fairly widespread. A driver is now available for Te χ which allows previewing of output on ascii devices, similar to what nroff provides. The capabilities of available drivers differ markedly, even for the same device, so it will pay to check them out before buying one. Troff drivers are discussed in more detail later in this paper.

2.3.5 Quality

The visual quality of a final document depends to a large extent on the author of the document, whichever formatter was used. However, in general, Te χ appears to do a better job than troff. For example, the Te χ line-breaking algorithm operates on an entire paragraph, whereas the troff algorithm only operates on a single line.

2.3.6 Variation

For a long while, there was only the one base version of troff, to which different sites added bug fixes and the inevitable enhancements. However, it would be a rare event if a single document printed differently at two different sites (except of course for font variations between printers). A larger source of variation was changes to macro packages, with documents variously relying on bugs, or the absence of them, with the same situation not necessarily guaranteed at a different site. Thankfully, most of the bugs have finally been weeded out of the macros, and if you use a standard macro package you should get the same result at any given site.

However, more recently completely new versions of troff have become available, which allow long names and a variety of other features, which is all well and good, but documents written for these versions will not print the same under the original version.

The situation with Te χ is that there are a plethora of versions, but each is required to pass "the trip test", which ensures conformity. If you choose to use the Computer

Modern fonts, you should get identical output, even on different printers.

2.3.7 Tables and Pictures

DWB provides quite good table formatting capabilities, and adequate picture formatting, with the `tbl` and `pic` preprocessors. Using the Table and Picture environment of LaTeX gives you about the same capabilities. Both formatters allow device dependant information to be passed through to the device although not all troff drivers allow this.

2.3.8 Security

This is not an issue many would normally associate with document formatting systems. However, certain troff commands can request the execution of external programs. While this is a very useful feature, it means that if you run a strange document through troff, you run the risk of anything happening. I am not sure whether TeX has this capability or not, but in any case you should be careful.

3. Output Devices

The large interest in Desktop Publishing which has suddenly arisen is undoubtedly due to the arrival of relatively high quality low cost output devices – laser printers. Described here is a summary of some of the devices available in Australia.

3.1 Marking engines

3.1.1 Introduction

All the printers here are laser printers, utilising basically the same technology, except for the Linotronics and Agfa-Gaevert, which are described later. Most laser printers are built around a "marking engine" of some sort, which is already available. Often there are photocopiers built around the same engines. It is these engines that determine the physical characteristics of the printer.

3.1.2 Physical Characteristics

3.1.2.1 Operation Basically, a laser scans a photosensitive drum, altering the charge on the drum. The drum then passes across toner, and picks it up, or doesn't, depending on the charge at that point on the drum. The toner is then deposited on paper, which goes under a fusing roller, so that the toner is melted onto the paper. The details of what charges go where are supremely unimportant, (in the canon engine, the drum and powder are both positively charged, with the laser discharging the areas where it strikes), however one aspect of the mechanism is relevant. Some engines are "write-black", which means that the toner is picked up wherever the laser has scanned the drum, whilst others are "write-white" which means that toner is picked up everywhere except where the laser has scanned the drum. There are two main results of this:

1. Because of charge leakage problems, a write-black engine does not produce very good large areas of black – they tend to be washed out. Write-white engines produce much better large black areas. The effect is not noticeable for normal text.
2. For engines of the same basic resolution, a write-white engine produces finer lines than a write-black engine. At 300 dpi resolution, a line only 1 dot wide should ideally be $1/300$ inch wide, but on a write-white machine it will be about 0.8 of this width, while on a write-black machine it will be about 1.2 times this width.

There can be a noticeable difference between some fonts on the two different types of printer. For example, some metafont definitions need to know whether their output will be on a write-white or write-black printer.

3.1.2.2 Resolution All of the laser-printer marking engines described here provide a 300 dpi (dots per inch) resolution. To get an idea of device resolutions, note that Group III Fax (commonest) has a resolution of 200 dpi vertical \times 100-400 dpi horizontal and Group IV Fax has a resolution of 400 dpi.

I have heard (rumour only) that an upgrade for canon engines is under development, whereby the photodiode that produces the laser is replaced, to increase the possible resolution to 400 or even 600 dpi. However, apparently the main problem is that the fine grade of toner that is required for such resolution leaks out of existing seals.

3.1.2.3 Speed The major difference between engines is the speed, measured in pages per minute, that the engine is able to print. This number is usually built into the model number of laser printers, perhaps with a couple of zeros tacked on the end. It is important to note that the maximum physical speed of the engine usually bears little relationship to the actual working speed of the printer. The limiting factor is the time taken by the controller to process the image for a page. Only very simple pages, or subsequent copies of a page, will be produced at the rated engine speed.

3.1.2.4 Toner Some engines use a replaceable toner cartridge, which also includes the print drum. This means that replacing the cartridge is easy, and you are ensured of consistently high print quality throughout the lifespan of the engine. But the cartridges are typically quite expensive, which means a higher running cost than a comparable printer with a fixed drum. Also, the fixed drum will be of a much higher quality to start with than the one in a cartridge, but in my experience this just means longer lasting, rather than producing better print quality.

In the US, various organisations will refill your toner cartridge for a fee. Apparently this can be done up to two times before quality becomes totally unacceptable. Discussion with various animals leads me to believe that it can be done yourself if you know how, and are VERY careful. Also, it is essential to get the right type of plastic toner, or you may clog your engine altogether (remember the *Imagen* saga).

I have heard (rumour only) that toner cartridges for laser printers are exactly the same as cartridges for photocopiers utilising the same type of engine, except that something was deliberately done by the manufacturers (a plastic lug added or removed or something) so that you can't use the cheaper photocopier cartridges in your laser printer.

3.1.2.5 Duty Cycle and Engine Life An important point to consider when choosing an engine is the lifetime of various components, and the rated duty cycle of the printer. Running the printer at a larger number of pages per month than it is rated for may cause problems. All printers will need to have their engines replaced after a certain lifetime. Printers which have fixed print drums will need to have them replaced after a certain period.

3.1.2.6 Page Size The circumference of the print drum is in all cases less than the typical page length. Page width is limited by the engine, but page length is typically limited only by the controller (esp. memory capacity). All engines except those used in

phototypesetters feed cut-sheet paper, and most are limited to A4 (or US letter) size, though some can print A3.

3.1.2.7 Output page stacking The size of the output tray varies between engines. A small tray can be a problem if large jobs are being printed - excess paper may either fall on the floor, or cause a paper jam in the printer.

If pages stack face-up, then they will be ordered incorrectly and need to be reversed by hand, unless the software pre-reverses the pages beforehand. Page flippers are available for all printers on which they do not come as standard. Indeed on some engines, simply removing the output tray will allow the pages to curl over face-down, and also allow a much larger number of pages to be safely stacked. Experiment a bit before ordering one.

3.1.2.8 Input trays The number and capacity of input trays also vary between engines, though all engines that I know of have a manual feed slot, as well as at least one input tray. Optional additional input trays may be purchased for most engine types. Typically these feed sheets through the manual feed slot of the printer.

3.1.2.9 Colour All the printers described here are monochrome. However toner other than black is available, so it should be possible to produce colour pictures of a sort by using some clever software and passing a single sheet through the printer several times, using a different coloured toner each time.

I heard somewhere (probably netnews) that Seiko are coming out with the CH-5301 colour laser printer, with a resolution of 152 dpi. Seiko Australia don't seem to have heard of it.

3.1.3 Canon 8ppm engine (Canon LBP-CX laser-xerographic engine)

Probably the engine in the most widespread use.

- Write-black
- Resolution: 300 dpi
- Max page width: ~215 mm (A4)
- Duty cycle has been quoted between 1K and 5K pages/month, depending on who you ask. Apple recommend about 4K pages/month.
- Engine life estimated at between 100K and 300K pages (depends on who you ask).
- Uses a replaceable toner and drum cartridge. Cost is over A\$100 (often much over).
- used in: LaserWriter, LaserWriter plus, QMS-PS800 and many non-PostScript printers, such as the Impact, Canon, etc.
- input tray capacity: 100 sheets
- output tray capacity: 20 sheets

A large array of accessories are available.

UNSW Electrical and Civil Engineering have been running their Apples fairly hard for quite a while; both are well over 100K pages. They are both still quite adequate, though signs of their impending doom have been evidenced (occasional streaks not

related to the toner cartridge or separation belt).

3.1.4 Toshiba 26ppm engine

- Write-white.
- Replace drum every 100K pages.
- Used in: Dataproducts LZR 2665
- Resolution: 300 dpi
- Max page width: ~300mm (A3)
- Input tray capacity: upper – 500 sheets; lower – 250 sheets

Accessories available include face down output trays, 10-bin sorter/collaters and 1500-sheet feeders.

3.1.5 Ricoh 8ppm engine

- Resolution: 300 dpi
- Duty Cycle: 10K pages/month
- Rated engine life: 600K pages
- Stacks face down (so your job comes out correctly ordered)
- Input tray capacity: 250 sheets
- Max page width: ~215mm (A4)
- Uses toner cartridges similar (identical?) to the canon engines.
- Used in TI Omnilaser 2108

3.1.6 Ricoh 15ppm engine

As for the 8ppm, except:

- Rumoured to be a "second generation" engine.
- Duty Cycle: 25K pages/month
- Rated engine life: 1.5M pages
- Input tray capacity: two, each holding 250 sheets
- Used in TI Omnilaser 2115

3.1.7 Xerox 12, 24ppm engines

- Resolution: 300 dpi
- Used in QMS-PS1200/2400.

3.2 Printer Control Languages

Most early laser printers adopted the same sort of printer control language as existing line-printers. This is mainly text, with some sort of escape sequence used to communicate control information to the printer.

More recently, genuine languages are being used to control printers. A "program" is written in this language and sent to the printer, which interprets the program, causing a page to be printed. These languages are known generically as "Page Description Languages" (PDLs). The very early precursors to these were the graphics languages that some devices accepted, and languages such as Impress, used by Imagen laser printers. The first of the higher level PDLs was "Interpress" from Xerox.

3.2.1 PostScript

More recently, and by far the most widely used high level PDL (at present), is PostScript from Adobe Systems. As well as being a very high level page description language, PostScript includes many features from general purpose programming languages, including iteration, recursion, and typed data structures. PostScript is a stack-oriented postfix language, incorporating a graphics model which allows for arbitrary shapes, painting (including shading and colour), fully integrated text and graphics, raster images, and general coordinate system transformations. The complete PostScript language (except for colour) has been implemented on a large number of high quality output devices. The first of these was the original Apple LaserWriter, which was remarkably bug-free considering the complexity of the task. What impresses me, is that what bugs existed were documented, with fixes or work-arounds readily available. Recent releases of PostScript have very few bugs, and most of these are in fairly esoteric areas.

When PostScript was released, Xerox put Interpress in the public domain, to increase acceptance of the language. While this has helped, and some major companies are supporting Interpress, this is often as well as other languages, such as PostScript.

3.2.2 C-Script

An american company called "Control-C Software" (Portland, Oregon) is developing a language called "C-Script" (it is written in C) which claims to be upward compatible with PostScript, but faster and better. They are talking with producers of laser printers, display cards, dot matrix printers and controller boards for laser printers. Apparently licensing will be much cheaper than PostScript is from Adobe. [This was summarised from the September 1986 issue of "Desktop Publishing"].

3.2.3 DDL

What most people see as a real competitor to PostScript is DDL (Data Decsription Language), from Imagen. Like PostScript, it is a genuine high level page description language, which is interpreted. It also is a stack-oriented postfix language. Some of the advantages claimed for DDL over PostScript are that DDL supports: both binary and ascii representations (gaining a size and transmission time advantage over the ascii-only PostScript); intelligent scaling of bit-map characters (presumably they do some sort of edge detection and conversion into an outline format); linear and non-linear scaling of fonts (PostScript allows only linear scaling); composite data objects (I don't know what is meant here, because a PostScript dictionary is certainly a composite data object); full object caching (PostScript only supports caching of fonts); document layout (this can be done in PostScript, but in DDL it is a specific part of the language).

The major backers of DDL are Imagen and HP. DDL has a long way to catch up in terms of acceptability compared to PostScript, but if it's good enough and (more to the point) pushed by enough major manufacturers, it just might do it. HP sell a DDL card

that slots into an IBM PC and connects to an ordinary HP LaserJet+ printer. Seems to me like the wrong way to go, but after all there are an awful lot of both IBM PCs and HP LaserJet+ printers.

3.2.4 Drivers

Although programs written in high level PDLs can be generated by hand, this is usually only done for special applications, or to get a specific effect. Normally the program will be generated by a driver associated with whatever word processing or document formatting package is already in use. Drivers for PostScript are available for the major Unix document formatters. A driver for Scribe is available from Unilogic. Drivers for troff and Te χ are available from a number of sources. For Te χ , there is dvi2ps (which comes packaged with most versions of Te χ), and other similar drivers claimed to be better, from other sources. The most common package in use with troff is TranScript, from Adobe Systems, for US\$1750. Competing products are available in the US. The only alternative (as far as I know) in Australia, is a driver available from Softway. This includes such features as the ability to download external fonts, whether in raster or PostScript format, full font memory management (so you can use an unlimited number of downloaded fonts on a page) without requiring to talk directly to the printer, the ability to include externally produced PostScript (for example a diagram drawn on a Mac), and the ability to use virtual page sizes (for example 4 virtual pages on a single real page, with some optionally rotated to produce a booklet format). Also, the Softway driver costs A\$1000, or US\$800.

3.3 Printers

A printer basically consists of a marking engine and a controller board, which interprets the printer control language and does all the necessary image processing. In the case of PostScript printers, the controller boards are almost without exception developed by Adobe Systems. I have heard that the QMS PS-1200 and 2400 were developed by QMS themselves, and that they were having some problems.

Adobe have made substantial improvements to both their controllers and their firmware since the early ones. The most significant change to the firmware is evidenced between the Apple Revision 1 (PostScript version 23.0) and Revision 2 (PostScript version 38.0). The later version of PostScript includes additional features, allows handshake interpretation of DSR/DTR signals, as an alternative to XON/XOFF, allows transmission rates higher than 9600 baud, allows overlapping of page imaging with execution of the next page and is substantially faster overall. A document describing the changes was distributed over the net from Adobe and is available from Softway. The format is that of an update to the "PostScript Language Reference Manual".

Most PostScript printers come "standard" with the following Adobe fonts: Times, Helvetica and Courier (constant width) each in Roman, Italic (or at least oblique), Bold and Bold-Italic, as well as a symbol font.

All the PostScript printers described here unless otherwise mentioned have both an RS-232 serial interface and an Appletalk (RS-422) interface, and provide a Diablo 630 emulation capability. Other interfaces and emulations are noted below where they occur.

3.3.1 Apple LaserWriter

Certainly the most widespread of postscript devices. Uses the Canon 8ppm engine. Originally, used PostScript version 23.0, which had some (mostly documented) bugs and limitations. If you use this, make sure you have a copy of the software patch that fixes many of the bugs, and also a copy of the a4 page definition routine. Both of these were distributed over the net, and both are available from Softway. The LaserWriter includes the standard set of Adobe fonts. There are many satisfied LaserWriter customers, with very few complaints. However, if at all possible, you should try to get a Revision 2 LaserWriter, which is just a PROM upgrade to PostScript version 38.0. Printing from troff happens at about 5 ppm on a revision 1 printer, but very close to the full 8 ppm on a revision 2 printer. An Apple LaserWriter costs about A\$8000 (inc. tax). Shopping around is advised.

3.3.2 Apple LaserWriter Plus

This is solely a PROM upgrade to an ordinary LaserWriter. It gives you a Revision 2 LaserWriter (ie PostScript version 38.0) and a lot of extra resident fonts: Palatino, New Century Schoolbook, Avant Garde, Bookman, Helvetica Narrow and New Century Schoolbook, all in Roman (or Book, or Light), Bold (or Demi), Italic (or Oblique) and Bold-Italic (or whatever), as well as Zapf Chancery Medium Italic, and Zapf Dingbats. The upgrade to an ordinary Apple LaserWriter costs about A\$1000 (inc. tax). If you have a Revision 1, it may be good value to go for the upgrade, because you will then have a Revision 2 printer. If you already have a Revision 2 printer, decide if you want to pay that much just for the extra fonts.

Also note that it is very advisable to have this upgrade installed by a technician. The proms are VERY sensitive to static, and it is important to allow a burn-in period before using the printer. If a technician installs the upgrade, you don't have as many hassles if things go wrong.

3.3.3 Sun LaserWriter

This is just an Apple LaserWriter, but it costs somewhat more because it comes bundled with Adobe's Transcript package.

3.3.4 QMS PS-800

Very similar to the Apple LaserWriter Revision 2. Uses PostScript version 38.0 on a Canon engine with 1.4Mb RAM. These printers are marketed in Australia by Bell and Howell, and retail for A\$11 034 + A\$1766 tax.

3.3.5 QMS PS-1200, 2400

Use the xerox 12ppm and 24ppm engines respectively. They are not available in Australia as yet, and I don't think a PostScript version of the 2400 is available anywhere.

3.3.6 Dataproducts LZR2660/65

Uses the Toshiba 26ppm engine (a write-white engine) and PostScript version 39.0. The 2660 handles normal sized pages (up to A4 or legal), while the 2665 handles A3. Cost of the 2665 is A\$30K.

When we tested this printer, we found a PostScript bug which shows up under certain conditions. Adobe have been notified and are working on it.

The controller it uses is similar to the Apple LaserWriter, revision 2, so despite the fast engine speed, the actual printing speed will be about 8 ppm (from troff) unless you are doing very simple pages, or multiple copies. The real win is in the A3 paper capability of the 2665. It is the only PostScript laser printer I have heard of that will handle paper this size. We printed some very nice larger-than-life digitised images on this printer.

3.3.7 Apollo Computer Domain/Laser-26

This is just the Dataproducts printer described above.

3.3.8 TI Omnilaser 2108

Uses the Ricoh 8ppm engine; has a page jogger (job separator); includes two font cartridge slots. It is a PostScript printer, also including additional emulations: HP LaserJet+, TI 855 DP/WP, HPGL (Hewlett Packard Graphics Language, used by most HP plotters). Includes additional centronics-type parallel interface. The cost is A\$9 990.

Given the higher duty cycle and longer life, this printer sounds like it could very well give the Apple and other canon-engine based printers a run for their money. The face-down stacking and larger paper bins that come as standard are also useful, although these can be had as accessories on other printers (though they are usually expensive).

Also, given the plethora of emulations and the centronics interface, just about anyone should be able to use this printer with existing software for either text or graphics (although not using the PostScript capabilities would be a waste).

3.3.9 TI Omnilaser 2115

This is the same as the 2108, but uses the Ricoh 15ppm engine, and has 3Mb RAM. The cost is A\$14 500.

Again, this sounds like good value for money for a faster printer.

3.3.10 DEC Printserver 40

Right at the top end of the laser printer market is this offering from DEC. It is a PostScript printer, driven by a vax II, and can print up to 40 ppm on sheets up to A3 in size. It has a duty cycle of 50K pages/month, 3 input bins, holding 2500 sheets each. It comes with 29 fonts (presumably the same ones as the LaserWriter Plus), and additionally emulates Ansi sixels, REGIS graphics (DEC's graphics language), and Tektronix 4010/4014. The interface is only via Ethernet, and the cost is A\$91 010, ex tax.

3.3.11 Laser Connection PS Jet hood

This is not a printer, but rather a PostScript controller board that can be fitted onto any existing laser printer that uses a Canon engine and has a top mounted controller card (such as the HP Laserjet, QMS Kiss, and presumably Impact). The board just replaces any existing controller board, to give you a PostScript printer with 2Mb RAM. The cost is A\$6250 ex tax, so you wouldn't go out and buy a cheapo printer with the upgrade in mind, but you might upgrade a non-PostScript printer if you already had one, and didn't want to buy a new PostScript printer. Laser Connection is a subsidiary of QMS. The product is available in Australia from Megavision.

3.3.12 Agfa-Gaevert P400-PS

This PostScript printer uses LED array electro-photographic imaging, rather than the more conventional laser scan, and has a resolution of 406 dpi. The controller is Adobe's new 68020 based 'Atlas' controller, which includes 6Mb RAM – a 1Mb font cache (by far the slowest part of printing on PostScript printers is caching characters), two 2Mb page buffers, 1Mb user VM – and a 20 Mb hard disk. The engine speed is 16-18ppm and apparently the controller usually manages to drive it that fast. Comes with two paper bins, one of which holds 2000 sheets, and a face-down output stacker. Interfaces additionally include a centronics-type parallel connection.

Sounds like a really nice machine, but unfortunately Agfa-Gaevert in Australia say they don't even have pre-release information on it, and that it will probably arrive in Australia the middle of 1987. The cost is ~180K Francs (US\$28K).

Note that the P400-PS, although using the same marking technology as the earlier P400, has a completely different controller. Apparently the previous command language was horrible and full of bugs. They won't even upgrade a P400 to a P400-PS because there are too many differences. Also, the P400-PS is cheaper than the P400.

Delairco Linotype Linotronic series

These are publication quality PostScript devices. The Linotronic machines are not so much laser printers as phototypesetters. Linotype advertise them as "laser-setters", which is an apt name, because a scanning laser beam images directly onto photographic film, or light-sensitive paper. They feed from continuous rolls rather than cut sheets. Available as options are hard disks for font storage (the entire Mergenthaler font library is available on an 86 Mb Winchester), and high-speed options. I forget exactly which version of PostScript they use, but it is a recent one, around 39.0.

Bureau services are available which will print stuff on these machines from PostScript.

3.4.1 Linotype Linotronic 100

Resolution of up to 1270 dpi; additional centronics interface; standard PostScript fonts; 17Mb (formatted) hard disk. Print speed is up to 240 lines/minute. Cost is A\$69 250.

3.4.2 Linotype Linotronic 300

Resolution up to 2400 dpi; paper width up to 305 mm wide (A3); print speed up to 585 mm/minute. Cost is ~A\$96K.

3.4.3 Linotype Linotronic 500

Resolution up to ~1500 dpi; paper width up to 457 mm wide (newspaper size); print speed up to 1040 mm/minute. Cost is > A\$100K.

3.4.4 Impact

This is the cheapest laser printer available on the Australian market. It uses the Canon engine and is not a PostScript printer. When they were first released, they had so many bugs in the firmware that you'd be lucky to print plain text without problems. The manual was next to useless and there were lots of problems with the organisation – support was abysmal, and quite a few customers just gave up in the early days.

Since my first experience with them, I have talked to a salesman who told me that they had a major internal reorganisation, and support was much better, the manual has been reprinted and most of the bugs had been fixed. I believe this to be pretty much true, but there are still problems. Some of the design decisions leave a lot to be desired, there are still some bugs, and the manual still contains errors. If all that you want is a good quality printer, and you don't intend to try and do anything at all fancy with it, then it is probably your best bet.

However, make sure that whatever is generating output for the device has specifically been configured for an Impact. It claims to emulate an HP Laserjet and an Epson, but it doesn't have enough memory to do the job properly. Also, make sure whatever your software is, that you can change the default lead-in character from 'l' to something else (like Control-A, but *not* ESCAPE), or you'll never be able to print a table.

For a first try, the Impact is a nice machine – it even has an LCD readout, with a kangaroo that hops across the front. It will print plain text in portrait and landscape mode. But if you want to reliably do anything more than that, go to a PostScript printer.

4. Images

The availability of high quality output devices to the downtrodden masses of course opens up wider possibilities than just printing text. Laser printers can be used as fast plotters, so you can draw all sorts of fancy plots. Drivers are available for most graphics packages to drive PostScript or other devices. But an area that didn't really exist for most people before laser printers was that of printing bitmapped images.

Most laser printers have the capability of doing some sort of image raster downloading, but many only have enough memory to do a few square centimeters. All PostScript printers have enough memory to download a full page raster image.

A monochrome raster image is almost always represented on a computer as a 2-dimensional array of pixels. Each pixel has a value which represents the image intensity or "grey level" at that point.

The resolution of an image refers to the amount of visual information contained in the image. This is affected both by the range of values any given pixel may take on (most often measured as the number of bits used to store that value), and the number of pixels in the image.

Since a laser printer is only physically capable of displaying 1 bit pixels (ie a dot is either black or white – nothing in between) we can only truly print 1 bit of information per pixel of our image. However, because of the averaging capability of the human eye, we can get around this restriction quite easily by simulating grey levels with differing mixes of black and white for each pixel. The easiest way of doing this is to draw each pixel as a blob, the size of which varies according to the grey level it represents. This technique is the same as that used to print photographs in newspapers and is known as halftone screening. Of course, this means that the number of image pixels we can represent in a given area on the page is reduced, according to the number of possible grey levels each blob may have to represent.

The halftone screening process may be done by host software, or if you have a PostScript device it can be done on the device. PostScript allows not only the halftone screen frequency (ie the number of blobs/inch) to be adjusted, but also the spot function itself which controls the way the blobs are drawn. It also handles averaging of pixels where there are more pixels in the image than can be represented on the device.

To get a better idea of all these concepts, refer to Appendix 4 – the page with the 9 images of a face on it. Image 2 in the middle of the top row is the "normal" image, using the default parameters on the LaserWriter. Decreasing the halftone screen frequency results in image 1: a much larger range of grey levels are available, but the number of distinct pixels represented has been decreased. Conversely, increasing the screen frequency results in image 3: more distinct pixels can be represented, but fewer grey levels are available. Images 3, 4 and 5 show the variation with the number of grey levels available per pixel. The grey level used in these images is determined by a simple proportional cut-off, but this is by no means the only way. For example, the cut-off point for the 1-bit image (image 4) can be varied up or down to obtain different contrast levels, so that the final image could be darker or lighter than the one shown. Note that there is little difference between image 6 and image 2. The limiting factor is the 300 dpi resolution of the output device. Finally, the last row of images illustrates the differences in changing the number of pixels in the image.

5. Appendices

5.1 Documenter's Workbench

A sample of some of the capabilities of DWB on a PostScript device.

5.2 Fonts

A sample of some of the fonts available on PostScript devices using troff. Note the troff 2-letter naming scheme used: XX normal; Xx bold; xX italic (or oblique); xx bold italic. Some names that would seem to be more obvious than those given are reserved for the hershey fonts, which are not shown here. Printing this many downloaded fonts takes a *long* time. It is also a good test of a driver's font management capability – there is only enough memory to hold a few downloaded fonts at a time.

5.3 Star Chart

This is a sample output page from the starchart program that was recently posted over Usenet (generating pic commands).

5.4 Images

Samples of bitmapped image rasters.

- Different representations of the same image.
- Candidate for an AT&T T-Shirt.
- A satellite photo from Macquarie University, and a picture of the space shuttle.
- A screen dump from a Sun.

- Some politician. This file was posted over Usenet and produced using a fisheye optical transformation.

5.5 PostScript

Some samples of what can be done with PostScript:

- The UNSW machine room.
- Some excerpts from the PostScript tutorial book.
- The M.C. Escher square limits picture.

Documenter's Workbench on a PostScript™ device.

Documenter's Workbench (DWB) is a suite of programs which run on the UNIX[†] operating system. These include: TROFF, a document formatting program; TBL, a preprocessor for formatting tables; EQN, a preprocessor for typesetting equations; PIC, a preprocessor for producing diagrams; and GRAP, a preprocessor for producing graphs.

The preprocessors make it easy to produce quite complex tables, equations, diagrams, or graphs. The device independent output from TROFF is interpreted by a postprocessor, or driver, for any specific output device. Softway has developed such a driver that works with all PostScript devices. These include laser printers and phototypesetters manufactured by Apple, QMS, DataProducts, Texas Instruments, DEC, AGFA-Gaevert and Linotron, amongst others. These pages were produced on an Apple LaserWriter Plus.

Resident fonts on the Apple LaserWriter™ are: Times Roman, *Times Italic*, Times Bold, ***Times Bold Italic***, Helvetica, *Helvetica Oblique*, **Helvetica Bold**, ***Helvetica Bold Oblique***, Courier (Constant Width), Courier Oblique, Courier Bold and ***Courier Bold Oblique***.

Additionally available on the LaserWriter Plus are: Palatino Roman, **Palatino Bold**, *Palatino Italic*, **Palatino Bold Italic**, Avant Garde Book, *Avant Garde Book Oblique*, **Avant Garde Demi**, *Avant Garde Demi Oblique*, Bookman Light, *Bookman Light Italic*, **Bookman Demi**, *Bookman Demi Italic*, Helvetica Narrow, **Helvetica Narrow Bold**, *Helvetica Narrow Oblique*, *Helvetica Narrow Bold Oblique*, New Century Schoolbook Roman, **New Century Schoolbook Bold**, *New Century Schoolbook Italic*, **New Century Schoolbook Bold Italic** and *Zapf Chancery Medium ItalicTM*.

Any font may be printed in any size (eg **aCDE**), aspect ratio (eg **ABC**), or slant (eg **HAD**). Some resident fonts may be printed in outline (eg **Roman**, **Italic**, **Helvetica**,

TM PostScript is a trademark of Adobe Systems, Inc. © 1984 Adobe Systems, Inc.

[†] UNIX is a trademark of AT&T Bell Labs.

LaserWriter is a trademark of Apple Computer, Inc.

The typefaces: Helvetica, Times, Palatino and New Century Schoolbook are trademarks of Allied Corporation; ITC Avant Garde, ITC Bookman, ITC Dingbats and ITC Zapf Chancery are trademarks of International Typeface Corporation.

Avant Garde, and others). Also available are a large number of down-loadable fonts, such as *Old English*, chess (▲●■■...) and even faces (█.█.█.█). Any font in Berkeley vfont format, Documenter's WorkBench glyph format, the format in which the Hershey fonts were distributed in over UseNet, or any PostScript font definition may be used. Because Softway's driver handles font management, any number of fonts (downloaded or resident) may be printed on a single page, without exceeding memory limits on the printer – without needing to interrogate the printer.

Of course, troff allows all the normal document processing features, such as justification, hyphenation, underlining, [boxing] etc. all under user control. Powerful macro packages enable you to specify the overall style of a document (letter, memo, report etc.) and to use features such as automatic list generation, page numbering, table of contents generation, footnotes, etc.

All this can be done using TROFF alone.

TBL

The TBL preprocessor may be used to easily construct tables. The following is a simple example.

Devices that support PostScript				
Name	Speed	Resolution	Approx cost	Comments
Apple LaserWriter	8 ppm	300 dpi	A\$ 8K	Canon LBP-CX engine
QMS PS-800	8 ppm	300 dpi	A\$11K	Similar to Apple
QMS PS-1200, 2400	12, 24 ppm	300 dpi		Xerox engine
TI Omnilaser 2108	8 ppm	300 dpi	A\$ 9990	Ricoh engine; 250 sheet paper bin.
TI Omnilaser 2115	15 ppm	300 dpi	A\$ 14 500	Ricoh engine; 2 paper trays; 3Mb RAM.
DataProducts LZR 2665	26 ppm	300 dpi	A\$ 30K	Toshiba (write-white) engine; A3 page feed
DEC Printserver 40	40 ppm	300 dpi	A\$ 91K	
Agfa-Gaever	16 ppm	406 dpi	US\$ 28K	Adobe 68020 based 20Mb hard disk, 1Mb font cache, 2 × 2Mb memory, LED array electro-photographic imaging.
Linotype Linotronic 100	20-200 lpm	1270 dpi	A\$ 69 250	genuine phototypesetter
Linotype Linotronic 300	0.3-2 ppm	2400 dpi	~A\$ 96K	up to 305 mm wide
Linotype Linotronic 500	0.5-3.5 ppm	~1500dpi	>A\$ 100K	457mm wide

EQN

The following is a quote from the EQN User's Guide.

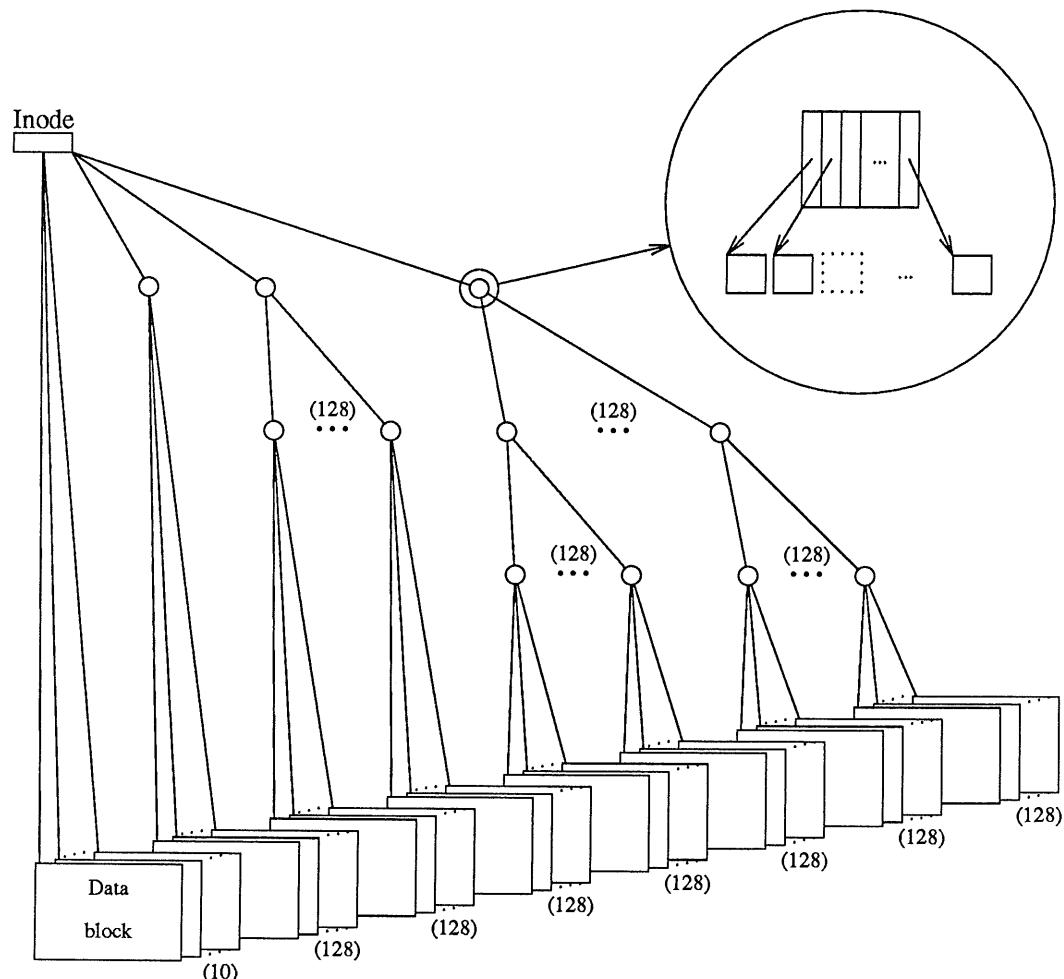
Mathematical expressions are described in a language designed to be easy to use by people who know neither mathematics nor typesetting. Enough of the language to set in-line expressions like $\lim_{x \rightarrow \pi/2} (\tan x)^{\sin 2x} = 1$ or display equations like

$$\begin{aligned} G(z) &= e^{\ln G(z)} = \exp \left(\sum_{k \geq 1} \frac{S_k z^k}{k} \right) = \prod_{k \geq 1} e^{S_k z^k / k} \\ &= \left[1 + S_1 z + \frac{S_1^2 z^2}{2!} + \dots \right] \left[1 + \frac{S_2 z^2}{2} + \frac{S_2^2 z^4}{2^2 \cdot 2!} + \dots \right] \dots \\ &= \sum_{m \geq 0} \left(\sum_{\substack{k_1, k_2, \dots, k_m \geq 0 \\ k_1 + 2k_2 + \dots + mk_m = m}} \frac{S_1^{k_1}}{1^{k_1} k_1!} \frac{S_2^{k_2}}{2^{k_2} k_2!} \dots \frac{S_m^{k_m}}{m^{k_m} k_m!} \right) z^m \end{aligned}$$

can be learned in an hour or so.

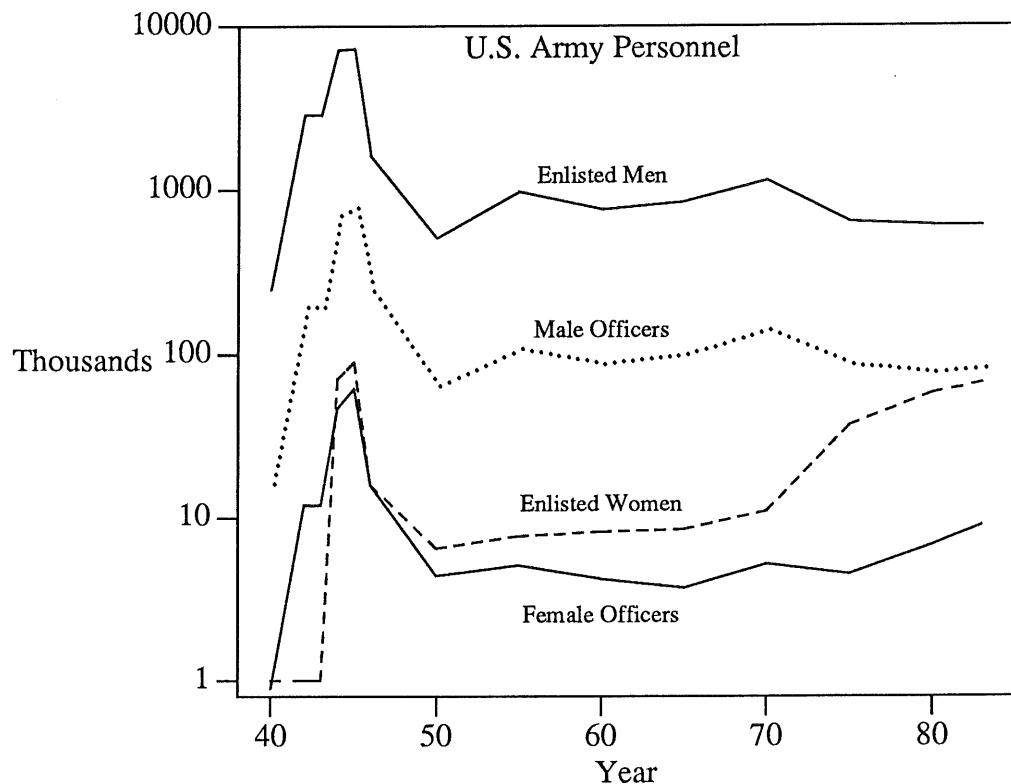
PIC

The following diagram, showing the UNIX file-system organisation is an example of the use of PIC.



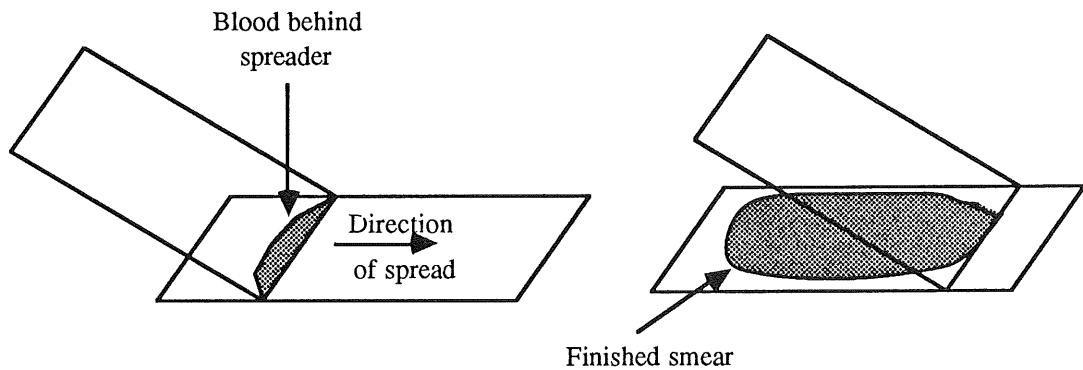
GRAP

Release 2.0 of DWB includes the GRAP preprocessor for easily drawing graphs of many different kinds. This is an example from the manual.



PostScript compatibility

PostScript produced from other sources can easily be included within TROFF documents. The following diagram was produced by MacDraw™ on a Macintosh™.



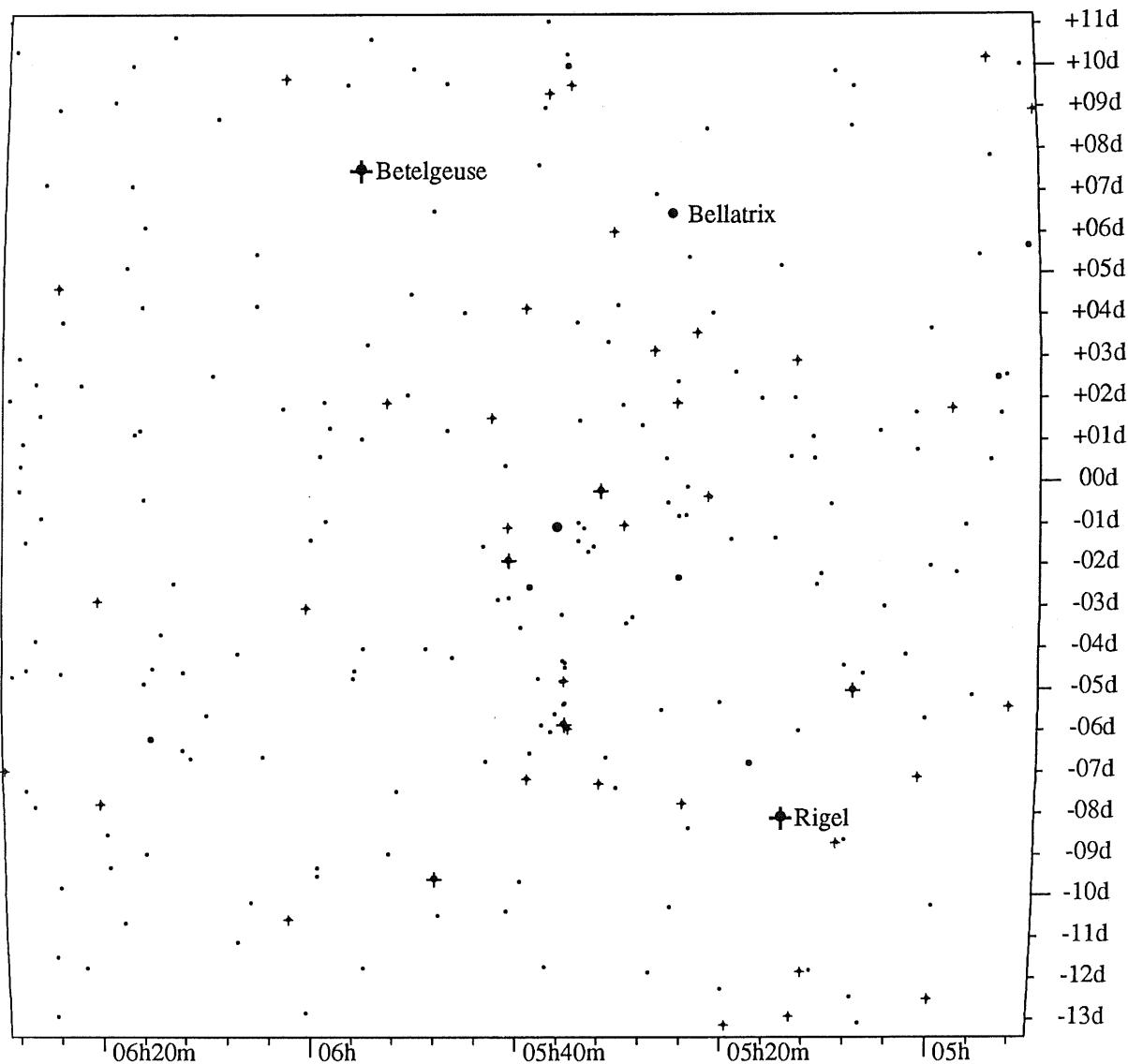
TM Macintosh is a trademark of McIntosh Laboratories, Inc.
MacDraw is a trademark of Apple Computer, Inc.

ft	Font Name	Sample Type
Apple LaserWriter		
R	Times-Roman	abegjmstwz ABEGJMSTWZ 3791
I	Times-Italic	<i>abegjmstwz ABEGJMSTWZ 3791</i>
B	Times-Bold	abegjmstwz ABEGJMSTWZ 3791
BI	Times-BoldItalic	<i>abegjmstwz ABEGJMSTWZ 3791</i>
CO	Courier	abegjmstwz ABEGJMSTWZ 3791
Co	Courier-Bold	abegjmstwz ABEGJMSTWZ 3791
cO	Courier-Oblique	<i>abegjmstwz ABEGJMSTWZ 3791</i>
co	Courier-BoldOblique	<i>abegjmstwz ABEGJMSTWZ 3791</i>
HE	Helvetica	abegjmstwz ABEGJMSTWZ 3791
He	Helvetica-Bold	abegjmstwz ABEGJMSTWZ 3791
hE	Helvetica-Oblique	<i>abegjmstwz ABEGJMSTWZ 3791</i>
he	Helvetica-BoldOblique	<i>abegjmstwz ABEGJMSTWZ 3791</i>
Apple LaserWriter Plus		
PA	Palatino-Roman	abegjmstwz ABEGJMSTWZ 3791
Pa	Palatino-Bold	abegjmstwz ABEGJMSTWZ 3791
pA	Palatino-Italic	<i>abegjmstwz ABEGJMSTWZ 3791</i>
pa	Palatino-BoldItalic	<i>abegjmstwz ABEGJMSTWZ 3791</i>
AG	AvantGarde-Book	abegjmstwz ABEGJMSTWZ 3791
Ag	AvantGarde-Demi	abegjmstwz ABEGJMSTWZ 3791
aG	AvantGarde-BookOblique	<i>abegjmstwz ABEGJMSTWZ 3791</i>
ag	AvantGarde-DemiOblique	<i>abegjmstwz ABEGJMSTWZ 3791</i>
BK	Bookman-Light	abegjmstwz ABEGJMSTWZ 3791
Bk	Bookman-Demi	abegjmstwz ABEGJMSTWZ 3791
bK	Bookman-LightItalic	<i>abegjmstwz ABEGJMSTWZ 3791</i>
bk	Bookman-DemiItalic	<i>abegjmstwz ABEGJMSTWZ 3791</i>
HN	Helvetica Narrow	abegjmstwz ABEGJMSTWZ 3791
Hn	Helvetica-Narrow-Bold	abegjmstwz ABEGJMSTWZ 3791
hN	Helvetica-Narrow-Oblique	<i>abegjmstwz ABEGJMSTWZ 3791</i>
hn	Helvetica-Narrow-BoldOblique	<i>abegjmstwz ABEGJMSTWZ 3791</i>
NS	NewCenturySchlbk-Roman	abegjmstwz ABEGJMSTWZ 3791
Ns	NewCenturySchlbk-Bold	abegjmstwz ABEGJMSTWZ 3791
nS	NewCenturySchlbk-Italic	<i>abegjmstwz ABEGJMSTWZ 3791</i>
ns	NewCenturySchlbk-BoldItalic	<i>abegjmstwz ABEGJMSTWZ 3791</i>
zC	ZapfChancery-MediumItalic	<i>abegjmstwz ABEGJMSTWZ 3791</i>
User Defined		
SU	Sydney University Crest, etc.	○●□■■■■■■
FA	People's faces (48x48x1)	█ █ █ █ █ █
Outlines		
Ro	Roman outline	abegjmstwz ABEGJMSTWZ 3791
Io	Italic outline	<i>abegjmstwz ABEGJMSTWZ 3791</i>
Ho	Helvetica outline	abegjmstwz ABEGJMSTWZ 3791
ho	Helvetica Oblique outline	abegjmstwz ABEGJMSTWZ 3791
Po	Palatino outline	abegjmstwz ABEGJMSTWZ 3791
po	Palatino Italic outline	abegjmstwz ABEGJMSTWZ 3791
Bo	Bookman outline	abegjmstwz ABEGJMSTWZ 3791
bo	Bookman Italic outline	<i>abegjmstwz ABEGJMSTWZ 3791</i>
Ao	Avant Garde outline	abegjmstwz ABEGJMSTWZ 3791
ao	Avant Garde Oblique outline	<i>abegjmstwz ABEGJMSTWZ 3791</i>
So	New Century Schoolbook outline	abegjmstwz ABEGJMSTWZ 3791
so	NCS Italic outline	<i>abegjmstwz ABEGJMSTWZ 3791</i>

Database: *Yale Star Catalog*

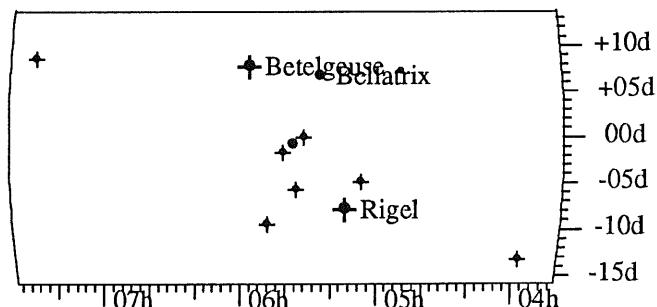
StarChart

Software: *AWPaeth@watCGL*



Orion and belt

=====
Right Ascension: 05h39m
Declination: -01d06m
Chart Scale: 24.700
+ <1.0 . <4.0
• <2.0 + <5.0
+ <3.0 . >5.0
=====

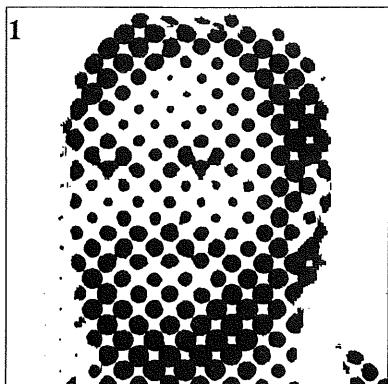


Device pixels/inch (screen frequency) for 85×128 pixel, 8 bit image

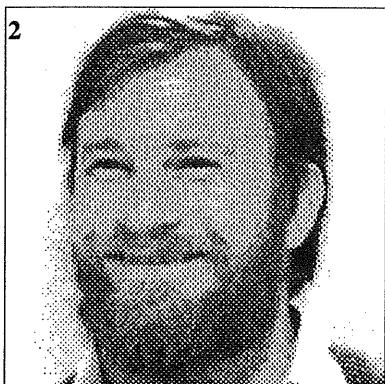
5

60

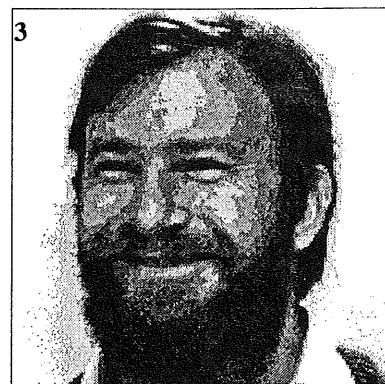
100



1



2



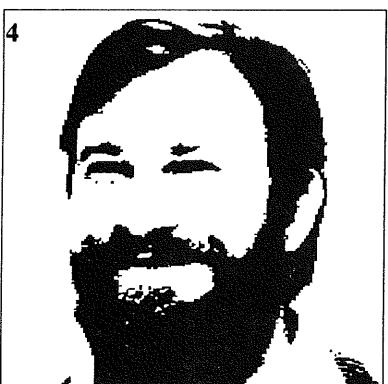
3

Image bits/pixel for 85×128 pixel image at 60 pixels/inch screen frequency

1

2

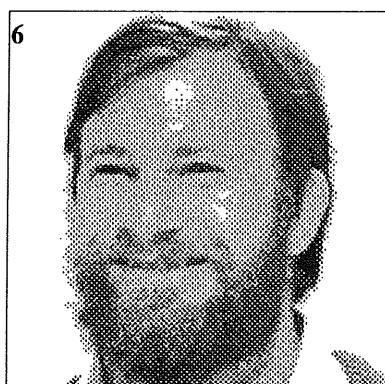
4



4



5



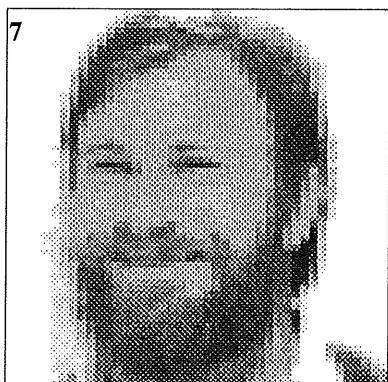
6

Image size (pixels) for 8 bit image at 60 pixels/inch screen frequency

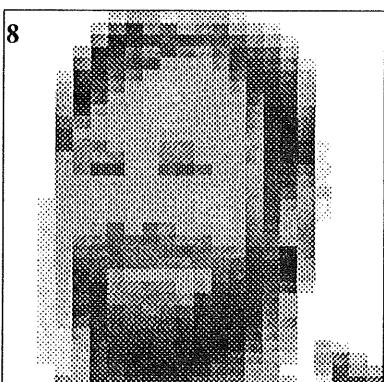
43×64

22×32

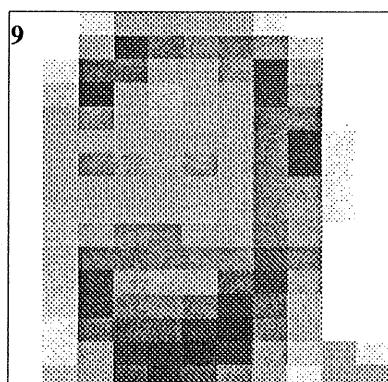
11×16



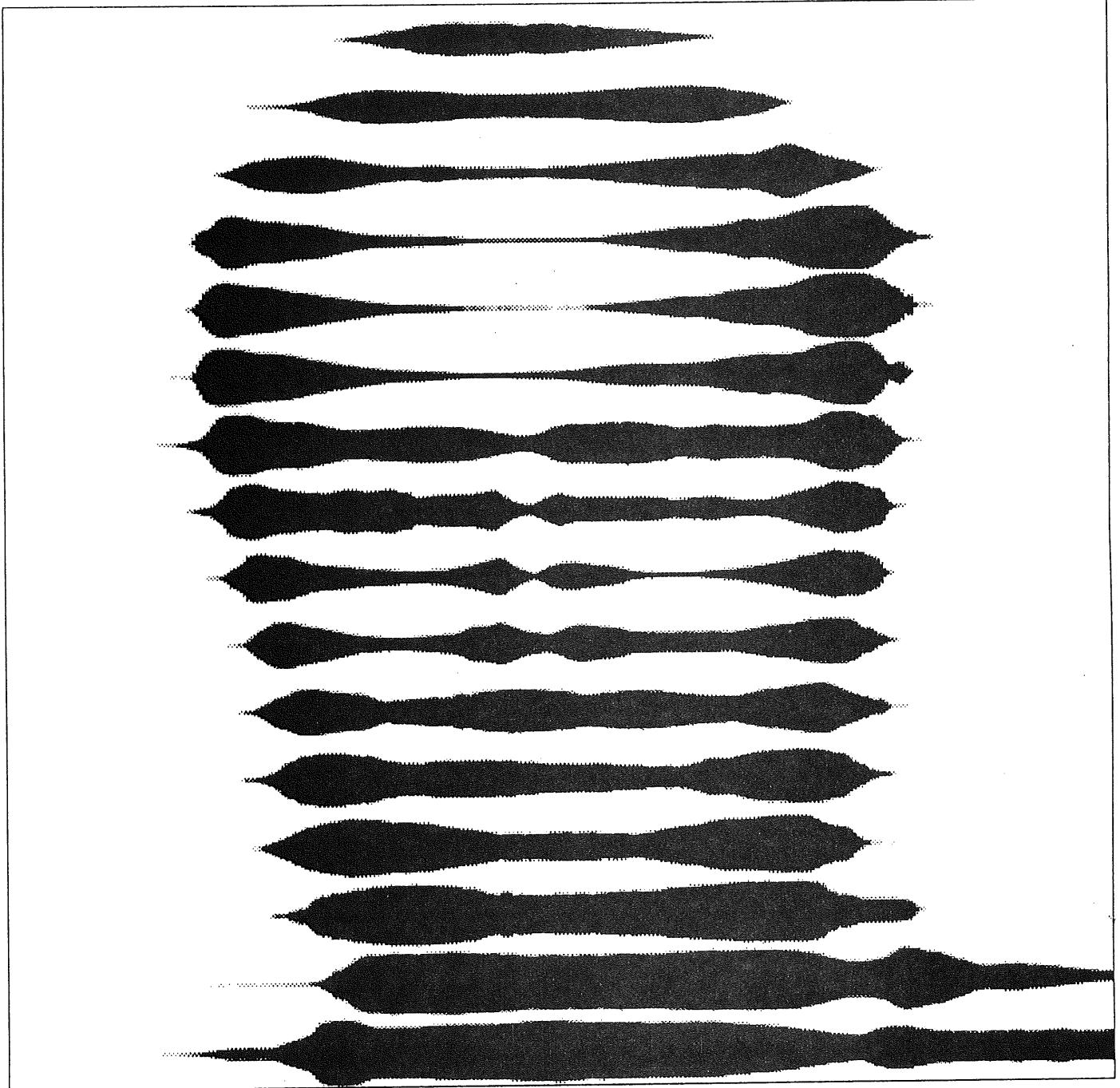
7



8

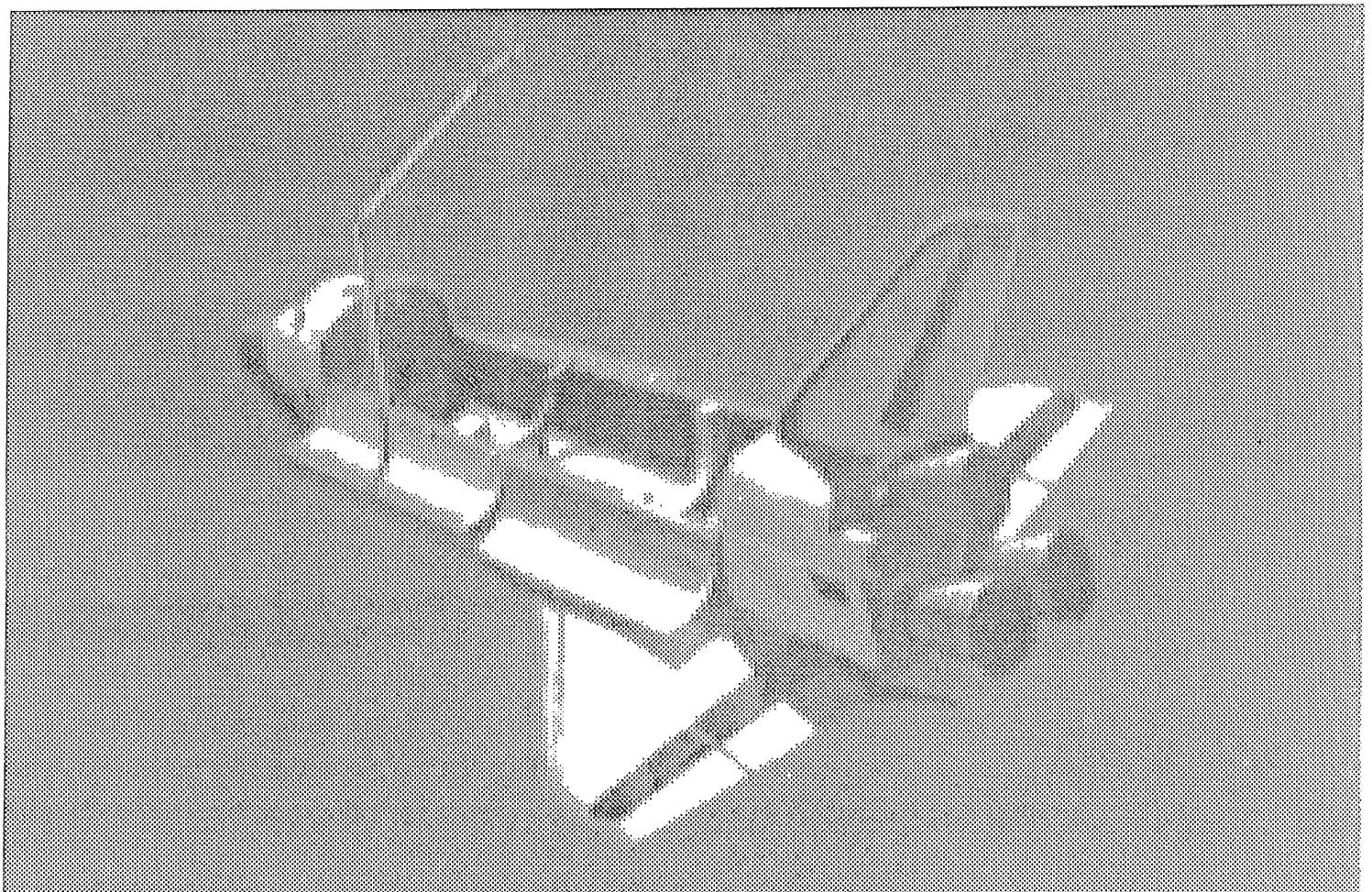


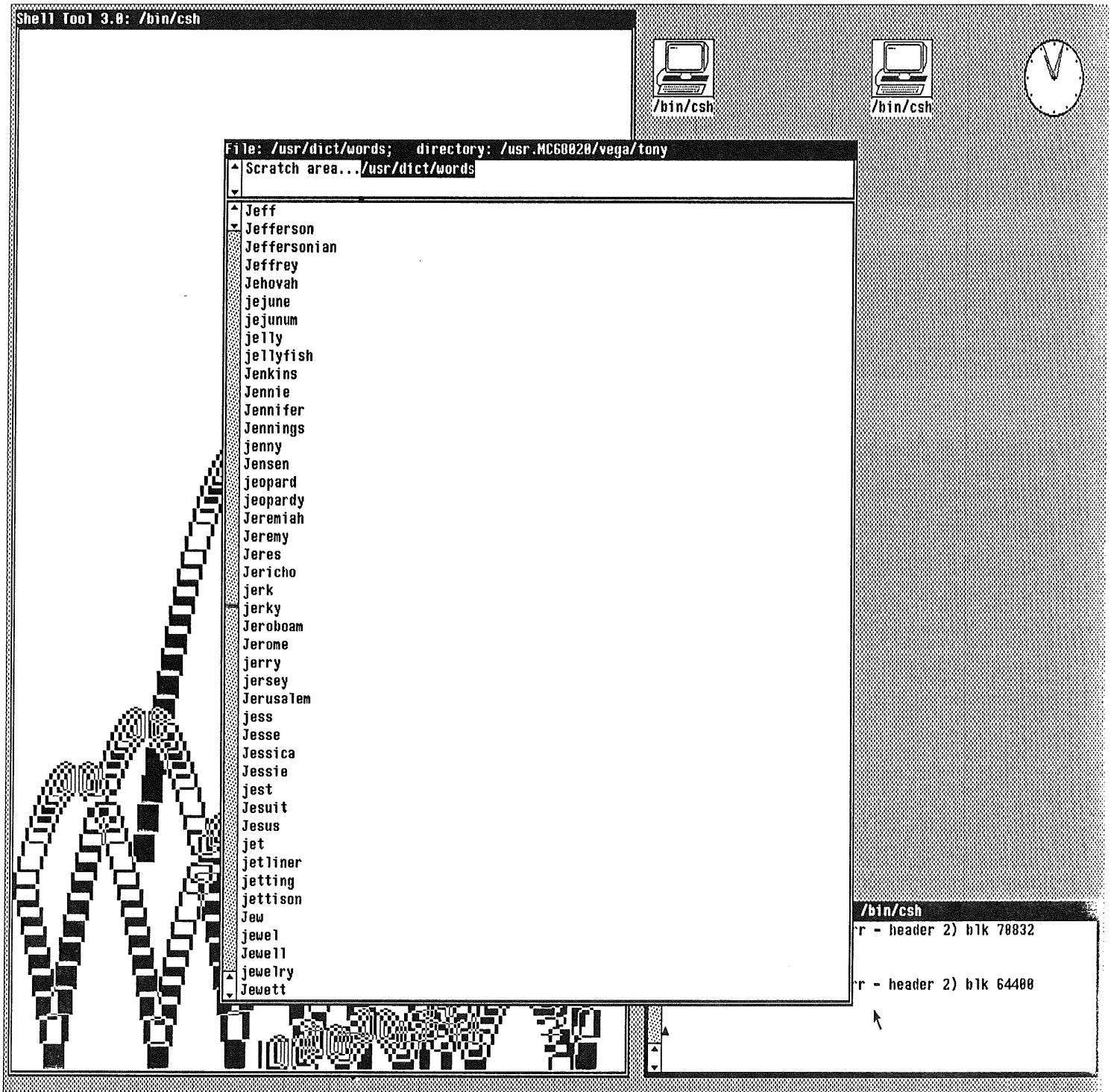
9



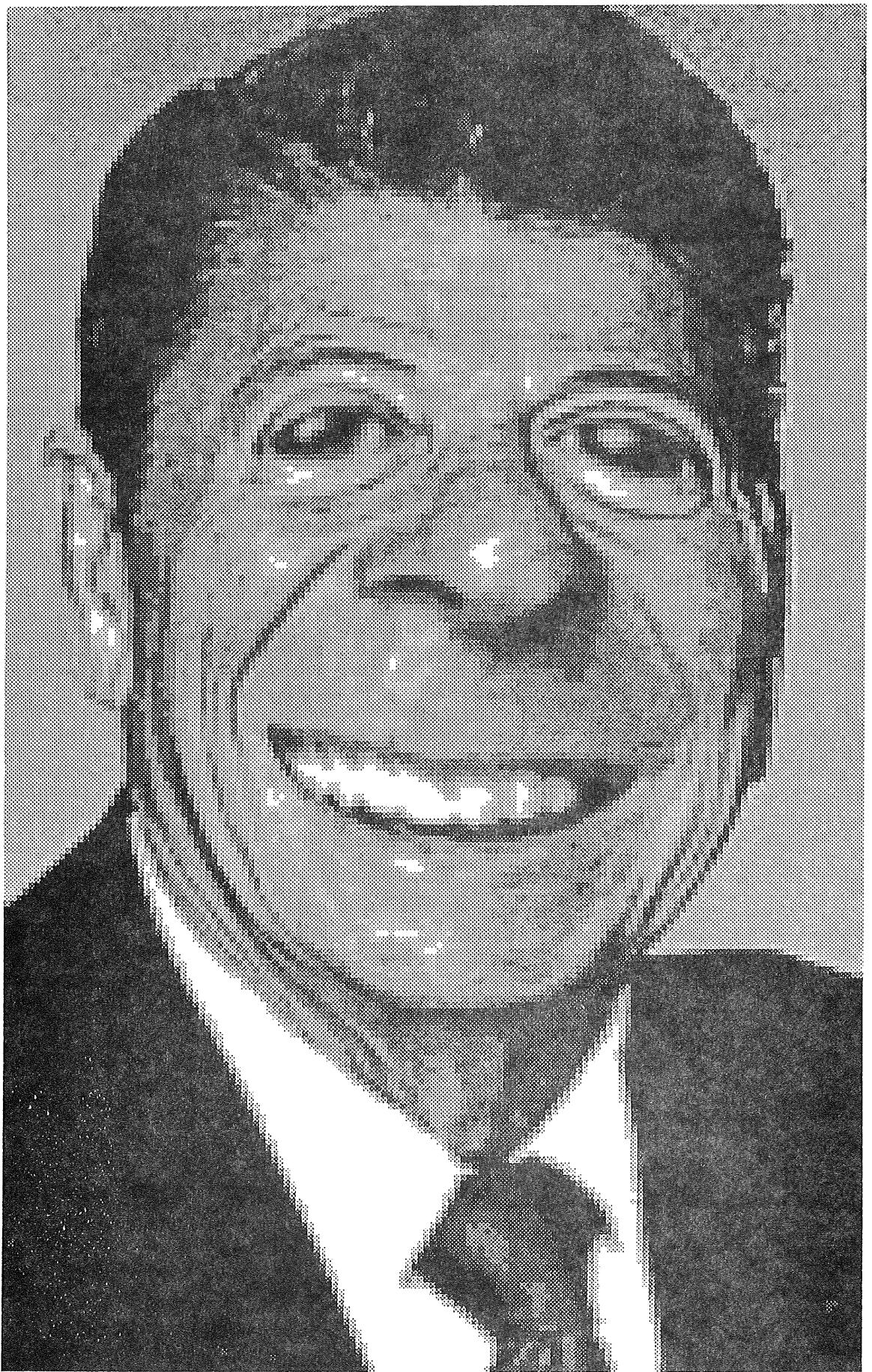
CHC-3 1981 SEP 10 1981

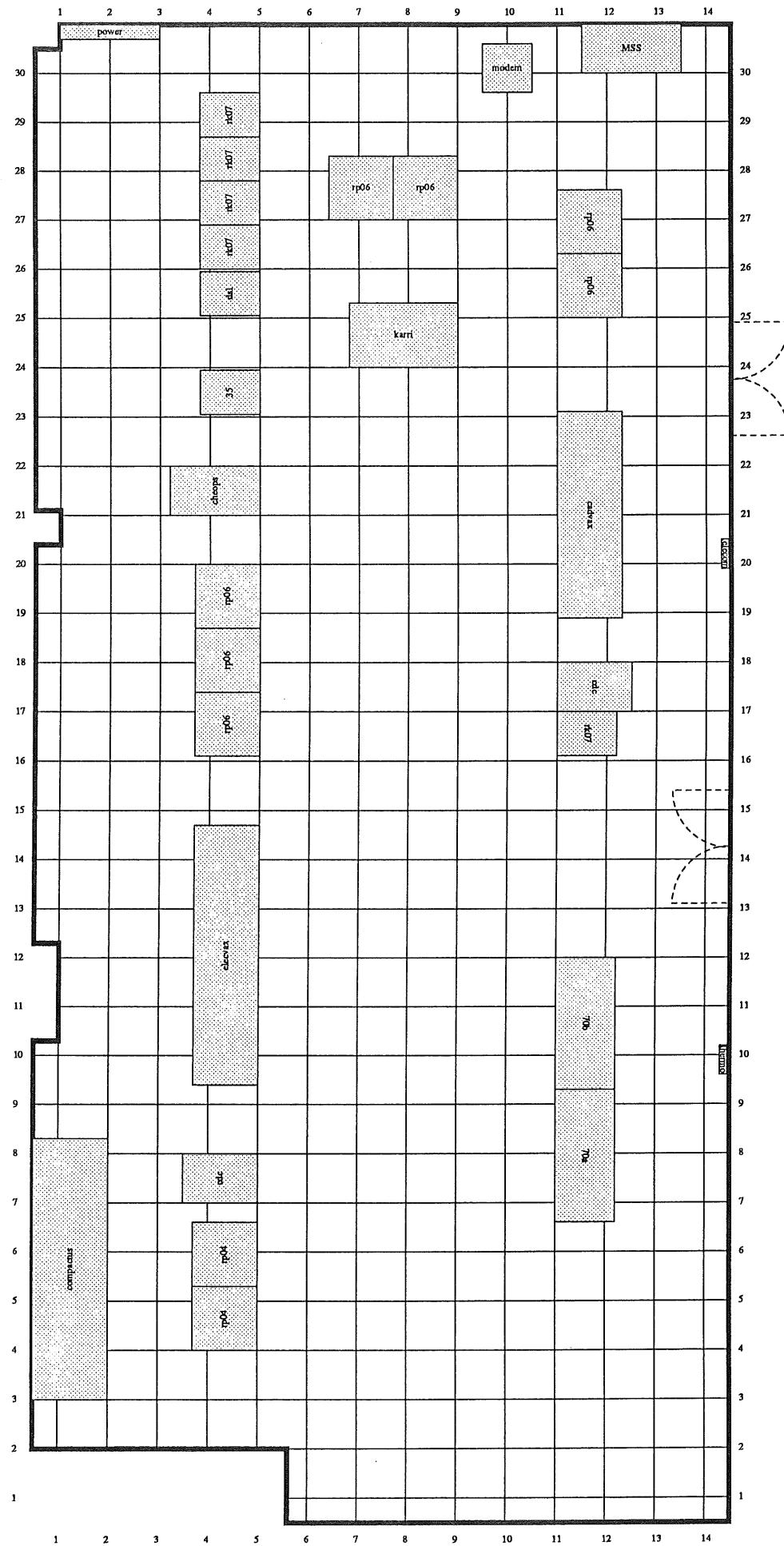






It Wash't me!

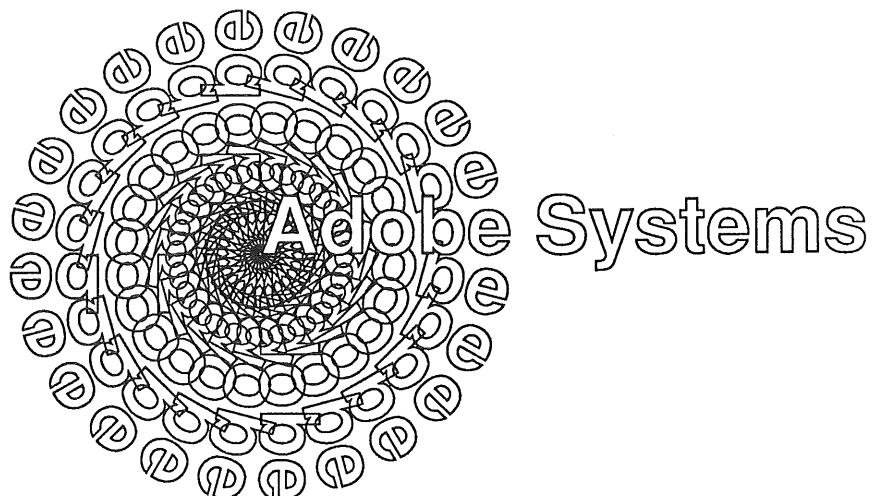




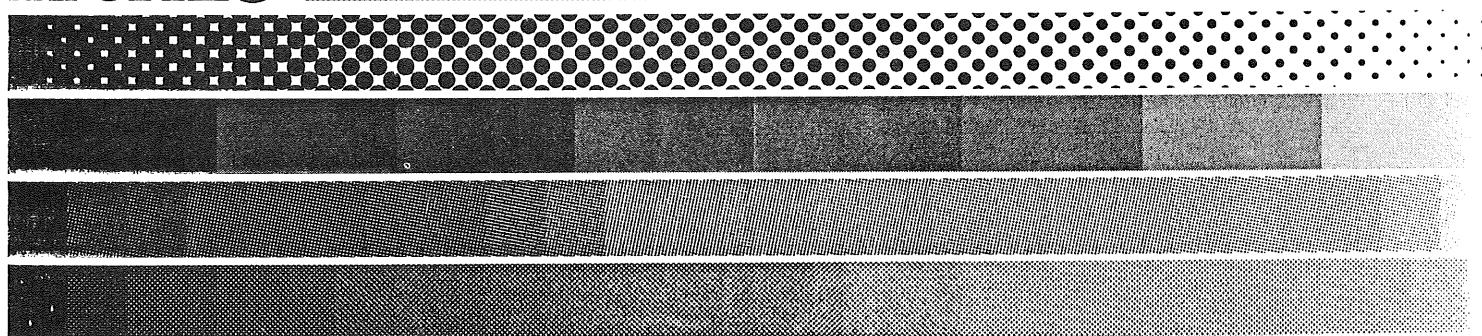
NewCenturySchlbk-Italic
Bookman-LightItalic
Helvetica
Courier-Bold
Helvetica-Narrow-Bold
Courier-BoldOblique
Times-Italic
Times-Bold
NewCenturySchlbk-Roman
Helvetica-Narrow-Oblique
Bookman-DemiItalic
Σψμβολ
Bookman-Demi
Helvetica-BoldOblique
Bookman-Light
Helvetica-Oblique
AvantGarde-Book
AvantGarde-DemiOblique
AvantGarde-BookOblique
AvantGarde-Demi
Helvetica-Narrow
ZapfChancery-MediumItalic
Courier-Oblique
NewCenturySchlbk-BoldItalic
Helvetica-Bold
Times-Roman
Times-BoldItalic
Helvetica-Narrow-BoldOblique
NewCenturySchlbk-Bold
Palatino-Bold
Courier
Palatino-Italic
Palatino-Roman
Palatino-BoldItalic
♦□❖*■*◎◆▼▲

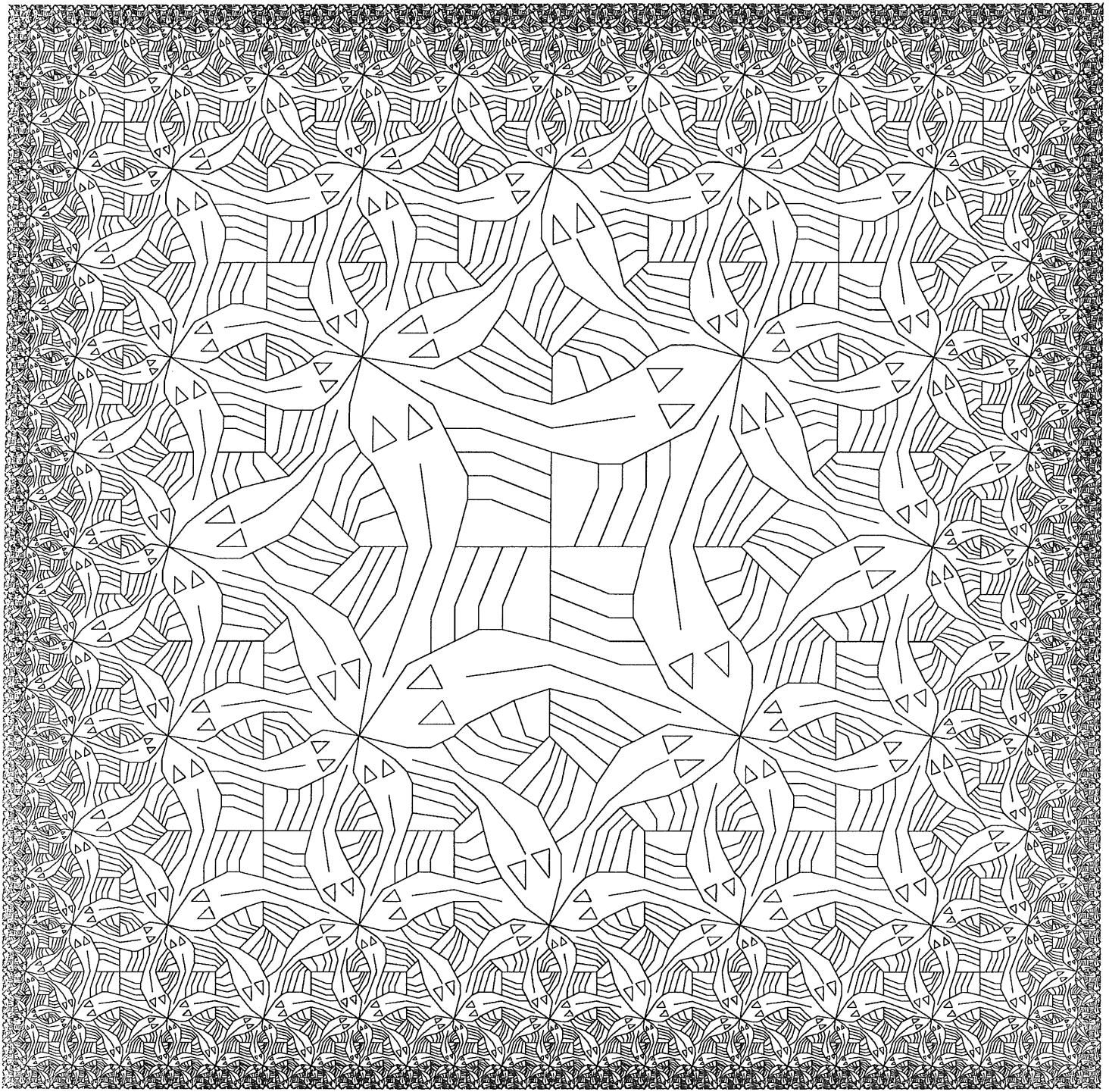
Shadow **Another**

StarLines



A B C D E F G H I J K L M N O P Q R





Newsletter Vol 6 No 2**Spring 1986**

How to NOT Win Friends and Influence People	1
Editorial or 'A letter from Canterbury'	2
Why Manchester?	3
Manchester Call for Papers	5
Unix Security	7
Awk Tawk	8
The Unix Hierarchy	11
RFS Architecture Overview	13
News From Finland - Unix and the Polar Bears	24
DKUUG since Paris 1985	26
Abstracts from the Florence Technical Programme	28
The Florence Contest	39
EUUG Tape Distributions	42

† UNIX is a Trademark of Bell Laboratories.

Copyright (c) 1985. This document may contain information covered by one or more licences, copyrights and non-disclosure agreements. Copying without fee is permitted provided that copies are not made or distributed for commercial advantage and credit to the source is given; abstracting with credit is permitted. All other circulation or reproduction is prohibited without the prior permission of the EUUG.

The UNIX Hierarchy

Compiled by Olle Johansson

Name	Description and Features
beginner	insecure with the concept of a terminal has yet to learn the basics of vi has not figured out how to get a directory still has trouble with typing <RETURN> after each line of input
novice	knows that 'ls' will produce a directory uses the editor but calls it 'vey' has heard of C, but never used it has had his first bad experience with rm is wondering how to read his mail is wondering why the person next to him seems to like UNIX so much
user	uses vi and nroff, but inexpertly has heard of regular expressions but never seen one has figured out that '-' precedes options has attempted to write a C program and has decided to stick with pascal is wondering how to move a directory thinks that dbx is a brand of stereo component knows how to read his mail and is wondering on how to read the news
knowledgeable user	uses nroff with no trouble, and is beginning to learn tbl and eqn uses grep to search for fixed strings has figured out that mv(1) will move directories has learned that 'learn' doesn't help somebody has shown him how to write C programs once used sed to do some text substitutions has seen dbx used but does not use it himself thinks that make is only for wimps
expert	uses sed when necessary uses macro's in vi, uses ex when necessary posts news at every possible opportunity write csh scripts occasionally write C programs using vi and compile with cc has figured out what '&&' and ' ' are for thinks that human history started with '!h'
hacker	uses sed and awk with comfort uses undocumented features of vi writes C code with 'cat >' and compiles with '!cc' uses adb because he doesn't trust source debuggers can answer questions about the user environment writes his own nroff macros to supplement standard ones writes scripts for Bourne shell (/bin/sh) knows how to install bug fixes

guru	<ul style="list-style-type: none"> uses m4 and lex with comfort writes assembly code with 'cat>' uses adb on the kernel while the system is loaded customizes utilities by patching the source reads device driver source with his breakfast can answer any unix question after a little thought uses make for anything that requires two or more distinct commands to achieve has learned how to breach security but no longer needs to try
wizard	<ul style="list-style-type: none"> writes device drivers with 'cat >' fixes bugs by patching the binaries can answer questions before you ask them writes his own troff macro packages is on a first-name basis with Ken, Dennis and Bill

[Reprinted from Svenska Unix Systems Användares Förening (EUUG-S) Medlemsblad 1/1986]

[This paper is published with the permission of AT&T. We regret that
the figures were not received in time for publishing. *The Editor*]

RFS Architectural Overview

*Andrew P. Rifkin
Michael P. Forbes
Richard L. Hamilton
Michael Sabrio
Suryakanta Shah
Kang Yueh*

AT&T
190 River Road
Summit, NJ 07901

ABSTRACT

Remote File Sharing (RFS) is one of the networking based features offered in AT&T's UNIX System V Release 3.0 (SVR3). RFS adds a new dimension to the user computing environment by providing transparent access to remote files. RFS allows access of all file types, including special devices and named pipes, in addition to allowing file and record locking on remote files. Careful attention to preserving the UNIX file system semantics ensures that existing binary applications can make use of network resources without failure.

By extending the notion of the UNIX mount, RFS allows a subtree of a server machine, to be logically added to the local file tree of a client machine. A message protocol based on the UNIX system call interface is used to communicate resource requests between the machines. The client and server machines employ a reliable virtual circuit style transport to transfer these messages. By adhering to a standard transport service interface accessed via the STREAMS [1] mechanism, RFS can operate over a wide variety of commercially available protocols without modification.

1. Introduction

The remote file sharing (RFS) feature of UNIX System V Release 3.0 (SVR3) is AT&T's offering to the distributed file system market. RFS provides the user with transparent access to remote files. Unlike other distributed file systems RFS preserves the full UNIX file system semantics and allows access to all types of files including special devices and named pipes. In addition RFS provides file and record locking for remote files.

This paper describes the set of goals on which RFS was based. An overview of the architecture used to achieve these goals is discussed, plus details of the RFS implementation in SVR3 is included.

2. RFS Goals

The main goal of RFS was to provide users and applications a means of accessing remote files. In the course of attaining this goal several subgoals were defined.

Transparent Access

The standard UNIX interface must be preserved. Access of a remote file must be the same as a local file. Accessing remote files must be independent of the files physical location.

Semantics

The UNIX System semantics must be preserved. All file types including special devices and named pipes must be accessible through RFS. File and record locking on remote files must be supported.

Binary Compatibility

Existing applications must not require modification or recompilation to make use of network resources.

Network Independence

In light of the rapid pace at which network products are evolving, it was decided that RFS should be cleanly separated from the underlying network. This allows for operating RFS, without modification, over a variety of networks that range from LAN's to large concatenated networks.

Portability

RFS code is largely machine independent to ease porting to different hardware environments. To simplify the integration of RFS into various UNIX Systems the changes to the kernel were localized and kept to a minimum.

Performance

Considering that a large cost of any remote operation is the network overhead, the performance goal was to minimize network access.

3. RFS Architecture

The RFS architecture is based on a client/server model using a central name server and a remote mounting scheme for connection establishment. Once connected the machines communicate using a message protocol based on the UNIX system call interface.

The STREAMS mechanism in conjunction with the transport service interface defined in System V is used to separate RFS from the underlying network, making RFS network independent.

By defining an RFS file system type, RFS is cleanly integrated into the UNIX kernel using the File System Switch (FSS) mechanism in SVR3.

To ensure security the normal UNIX file protection is extended to remote files and a mechanism is provided to map user id's.

3.1 Client/Server

A file sharing relationship consists of two machines, a client machine and a server machine. The file physically resides on the server machine, while the client machine remotely accesses the file. The client accesses the file by sending a request message to the server machine. The server provides the resource in the form of a response message to the client.

Any machine may be a client, server or both.

3.2 Connection Establishment

The RFS connection establishment involves locating and identifying a remote resource followed by remotely mounting it. The location and identification of resources is done using the RFS name server. The remote mount adds the remote file system to the local file tree. The remote mount model will be described followed by a discussion of the RFS name server.

Remote Mount

The remote mount model provides the same "tree building" approach used in UNIX. A client machine can add (mount) a remote file system from a server machine onto its local file tree. To support this a two step process is required. First, the server must (advertise) make a subtree of its local file tree available. Second, the client must add (mount) this subtree onto its local file tree (Figure 1).

Figure 1. Remote Mount Model

When a subtree is advertised a symbolic name is assigned to it by the server. In figure 1, the /usr subtree is assigned the name **USER**. A client machine now uses this symbolic name to mount this file system onto its local file tree.

Name Server

The RFS network should be viewed as a network of file systems rather than a network of machines sharing file systems. Therefore, it is necessary to logically separate a resource from its location. The RFS name server does just that.

The RFS name server maps resource names, which represent file trees that are available to share, into information about that resource. It allows machines to register the name of a resource, and it allows other machines to make queries about what resources are available. To accomplish this the name server maintains a centralized data base with a reliable recovery mechanism to avoid a single point of failure. This data base contains all currently advertised resources, mapping the symbolic name to the network location. The name server enforces uniqueness of symbolic names within a domain (see below) ensuring a consistent network view.

When a resource is advertised, the name server checks the symbolic name for uniqueness and if unique registers the resource in the data base. When the resource is mounted the name server converts the user specified symbolic name to the network location.

Domains

As the network grows, resource management becomes increasingly difficult. To alleviate this problem a domain based naming scheme is used. This concept allows machines to be logically grouped into a smaller, separate name space called a domain. For instance, all machines belonging to a single department in a large corporate structure may be partitioned into a single domain, carving the large corporate network into smaller more manageable pieces. Each of these domains have a central name server, guaranteeing unique resource naming within a domain.

To reference a resource within the local domain specifying the symbolic name is sufficient. To reference a resource from another domain it is necessary to prefix the symbolic name with the name of the domain in which the resource resides (Figure 2).

Figure 2. Domain Naming Scheme

3.3 RFS Message Protocol

The RFS message protocol is based on the UNIX system calls which are well defined^[2] and accepted. This protocol is used to communicate remote resource requests between client and server machines.

For each system call there exists a request and response message. The request message formats all pertinent information necessary to execute the system call, while the response message formats all possible results. The following brief description demonstrates the use of this protocol.

A client process, in the course of executing a system call encounters a remote resource. Execution of the system call is suspended, the clients environment data is copied into a request message, and the message is sent to the server machine. On the server machine, a server process services the request by recreating the clients environment based on the contents of the request message and executes the specified system call. The results of the system call are copied into a response message and sent to the client machine. Therefore, a remote system call requires only two messages a request message and a response message, thus minimizing network access.

3.4 RFS File System Type

The File System Switch (FSS) mechanism of UNIX SVR3 allows the same UNIX operating system to simultaneously support different file system implementations. Based on Peter Weinberger's (AT&T Bell Laboratories UNIX Research Laboratory) inode level switch, the FSS mechanism preserves system call compatibility while isolating different file system implementations from one another.

FSS separates the generic file system information from the file system specific information, and defines common interface between the kernel and the underlying file systems. This is done by dividing the inode into two parts, the file system independent portion (generic information) and the file system dependent portion. FSS now intervenes between the kernel and the file system by directing inode operations from the kernel to the file system specified by the "type" of the dependent inode.

RFS makes use of this feature by defining an RFS file system type. This file system type defines RFS type dependent inodes, which contain a communication pointer across the network to the "real" inode on the server machine (Figure 3).

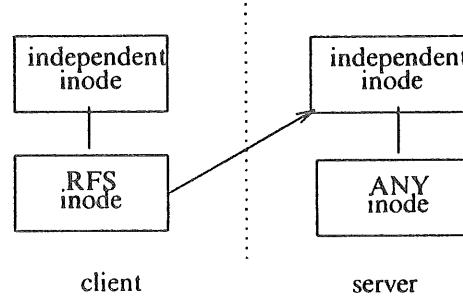


Figure 3. RFS File System Type

Now, inode operations resulting from a file descriptor based system calls are directed to the RFS module via the FSS. The RFS module in turn, uses the communication pointer in the RFS dependent inode to direct the request message.

3.5 Network Independence

By separating RFS from the underlying transport service, RFS is able to run over a variety of protocols and networks without modification. To accomplish this two problems had to be solved. First, it was necessary to choose what style of transport service RFS required. Second, a standard interface between RFS and the transport provider was needed.

Since RFS must work over a large concatenated network, and since the overhead of retransmission and error detection is high for datagram service over large networks, reliable virtual circuit service was chosen. To compensate for virtual circuit setup costs, a single virtual circuit is maintained between a client and server machine. This circuit is established during the first remote mount, and all subsequent mounts are multiplexed over this circuit. The circuit is held open for the duration of the mounts.

The second problem was solved using the transport interface (based on ISO Transport Service Definition) defined within the System V networking framework^[3] and the STREAMS mechanism. By adhering to the forementioned transport interface, a connection between RFS and a transport provider (also adhering to the transport interface) is then provided via the STREAMS mechanism. In addition, by allowing RFS to communicate over multiple STREAMS, RFS is able to make use of a variety of transport providers simultaneously (Figure 4).

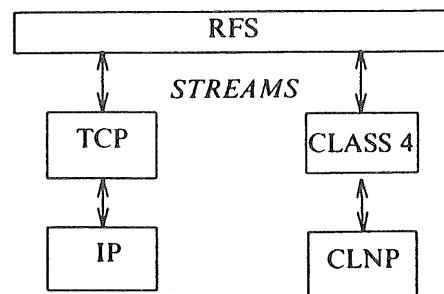


Figure 4. STREAMS Based Communication

3.6 Security

One of the problems in allowing one machine to transparently access files on another is the need to authorize the access. Two levels of security must be considered, the machine level and the user level. At the machine level a means of restricting clients from mounting a particular resource is provided. When a server machine advertises a resource the server can specify only those clients that are allowed to mount the resource, restricting all other clients. In addition, a server machine can be

configured to require a password check at the time a virtual circuit is established to the client machine.

At the user level the local UNIX security scheme has been extended to the network environment. In the local case access permission is based on a user's user identification (uid) and group identification (gid) compared to the file access rights. To make this scheme work for the remote case two problems must be solved. First, over a large network, a common password file (*/etc/passwd*) among a group of machines can not be guaranteed. Second, a means of restricting remote user access must be possible.

Both these problems are solved using a uid/gid mapping scheme. This scheme allows a uid/gid from a client machine to be mapped to a different uid/gid on the server machine. In the case of different password files, the uid/gid of a user on a client machine can be mapped to the uid/gid of that same user on the server machine (Figure 5).

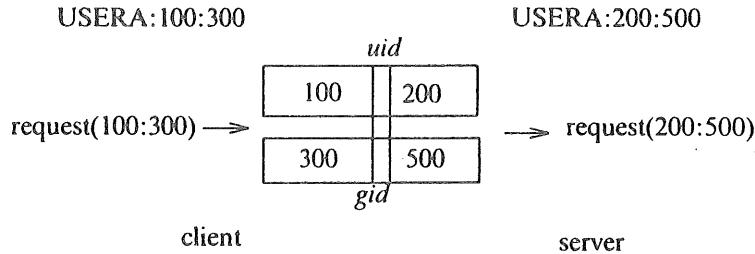


Figure 5. uid/gid mapping

Users can similarly be restricted on the server machine by mapping their uid/gid to an impotent value. Figure 6 shows how super user on a client machine is restricted from having super user privileges on the server machine.

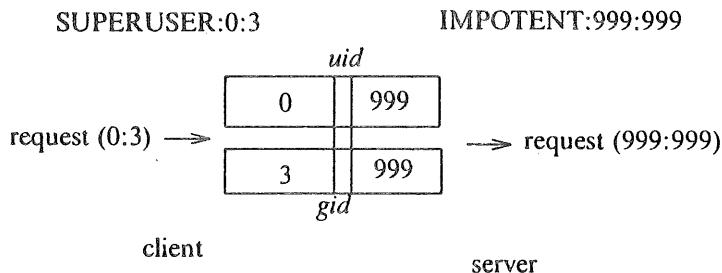


Figure 6. Super User Restriction

4. RFS Implementation

To minimize kernel change and to ease future porting, RFS was implemented as a separate module with a well defined interface into the kernel. The RFS module consists of approximately twenty source files and six header files.

The implementation was done in functional pieces. The *name server* was implemented separately, entirely at the user level. The *remote mount* code which establishes the data structures required to connect the client and server machines was separated from the remote access code. The remote access code was separated into *client* and *server* routines. This code makes up the bulk of the system. Finally, since RFS retains client state information on the server machine a *recovery* mechanism was implemented to resynchronize state information, in the event of a machine crash. The implementation details of these components will be discussed below.

Name Server

The RFS name server is a user level daemon that is initiated on each machine when RFS is started. Processes that want to communicate with the RFS name server open a stream pipe device that is

associated with the name server and use that stream pipe to send their requests. Communication between name servers on different machines uses the standard TLI to provide protocol independence.

The RFS name server is modeled as a transaction handler; it receives requests, performs an operation, generates a reply, and sends the reply to the originator of the request. It also acts as an agent of the requesting process if a request needs to go to a remote name server. A request first goes to the local name server daemon, which services the request if it can and otherwise forwards it to the appropriate name server. A non-recursive method is used for finding a name server that has the required information to avoid looping.

Remote Mount

The remote mount scheme extends the standard mount mechanism to include remote resources. As described earlier this is a two phase process where a server must advertise a resource before a client can mount it. The implementation required the addition of two new system calls, *advfs()* and *rmount()* and a new *adv* command plus modifications to the existing *mount* command.

The *adv* command in conjunction with the *advfs()* system call allows a server to advertise a subtree of its local file tree. The *adv* command interfaces with the name server to register the resource in the name server data base. The command then calls *advfs()*. The *advfs()* system call stores the symbolic name, and a pathname for the subtree (which is resolved to an inode), in a kernel advertise table.

The *mount* command has been extended to mount remote resources. The new *-d* option now specifies that a remote resource is to be mounted. For example:

```
mount -d USER /foo
```

requests that the remote resource associated with the name **USER** be mounted on the local mount point **/foo**. The name server is used to convert the specified symbolic name to the network location. If a virtual circuit does not currently exist between the client and server, the *mount* command sets one up. This will be discussed in detail below. The *mount* command then uses the *rmount()* system call to establish the kernel data structures required for the remote mount.

The *rmount()* system call takes the symbolic name, local mount point (pathname), and virtual circuit pointer as arguments. A remote rmount request which includes the symbolic name is sent to the server via the virtual circuit indicated by the virtual circuit pointer. The server uses the advertise table and symbolic name to locate the inode of the desired resource. If this client is authorized for this resource, the server records the client access in a kernel server mount table. Each time a client mounts a particular resource a entry is placed in this table indicating the clients system identification and the mount table index that the client used to record this mount. This information is used to resolve pathnames which traverse back out of a remote resource by using a **..** in the pathname. A response is then sent to the client containing a communication pointer to the inode of the advertised resource. Using this communication pointer the client creates an RFS inode. The RFS inode and the inode of the local mount point are then stored in the clients kernel mount table (Figure 7).

As mentioned before the *mount* command is used to set up a virtual circuit between the client and server machines. The *mount* command initiates a connection to a daemon (listener) process on the server machine using the standard Transport Level Interface (TLI) mechanism. Once connected negotiation of run time environment parameters is done. Included in this environment is release version numbers, security parameters, and hardware architecture types, to mention a few. If the machines have heterogeneous machine architectures, then external data representation (XDR)^[4] is used to allow these machines to communicate. Due to performance degradation XDR is only used when necessary. Once the negotiation is complete the virtual circuit is passed into kernel address space using a new *rfsys(FWFD)* system call. This virtual circuit establishment procedure moves the network connection complexity out of the kernel into user level routines.

Client

A client process is a process that accesses a remote resource. Remoteness detection is dependent on the type of system call being executed. For pathname based system calls, remoteness is detected

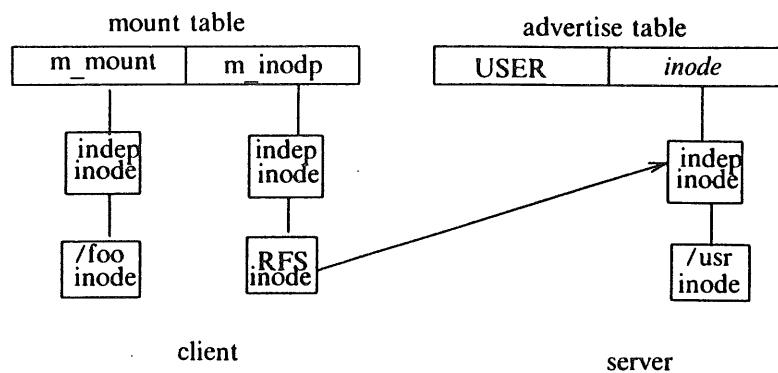


Figure 7. Remote Mount Data Structures

upon the traversal of a mount point of a remote resource (e.g., /foo in Figure 1). For file descriptor based system calls remoteness is detected when a RFS inode is encountered. The client implementation for these two cases is considered separately.

In the pathname case the system call uses the *namei()* and *iget()* functions to resolve the pathname to an inode. Specifically, *iget()* is used when traversing down into a mounted file system. The *iget()* function has been modified to pass control to the RFS module when traversing into a remote file system. Upon entering the RFS module the system call under execution is suspended. A request message representing the system call and the remainder of the pathname is sent to the server machine using the communication pointer from the RFS inode in the mount table. The client process then blocks until a response is received from the server.

For those system calls that require no further access of the inode (e.g., chmod, chown) control is returned directly (via longjmp) to the user, instead of through the system call routine which initiated the remote access. This allows RFS to do remote system calls without having to change each system call in the kernel, thus minimizing altered kernel code.

For those system calls which establish an inode which will subsequently be referenced (e.g., exec, open) control is returned to the initiating system call routine. Before control is returned an RFS inode is created using the communication pointer contained in the response message from the server. This RFS inode is then passed to the initiating system call routine.

The file descriptor case uses FSS for remoteness detection. A file descriptor for a remote file is the result of a pathname based system call (e.g., open, create) (Figure 8).

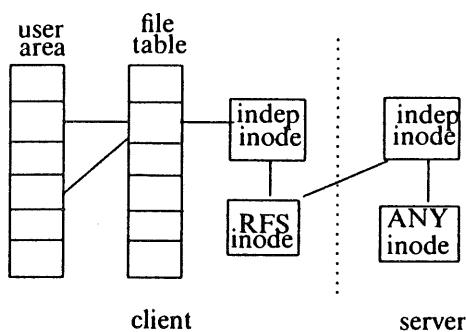


Figure 8. Open Remote File

The file descriptor is associated with an RFS inode through the local file table. In the course of executing the system call the FSS encounters the RFS inode. At this time control is then passed to the RFS module. The RFS module uses the same request/response mechanism described above to

complete the system call. Upon completion, control is returned to the initiating system call routine.

The *read()* and *write()* system calls require additional data transport. For the *read()* system call a read request message is sent to the server. The server in turn satisfies the request by sending all the requested data in the form of data messages to the client. For each data message received the client copies the data into the user supplied buffer. A light weight protocol between the server and the client is used to handle flow control. In an effort to minimize network access the last data message is combined with the response message for the initial read request.

The write system call behaves similarly but data messages flow from client to server. For each block of data required to satisfy the write request, the server sends a data request message to the client. The client responds by sending the next block of data to be written. The first block of data is combined with the initial write request message, saving a data request message and a data message, minimizing network access.

Server

The server is a kernel process, scheduled like any other process. It has a user area, although there is no user instruction space, no bss, and no text space. The code it executes is solely in the kernel so there are no context switches between user and kernel mode.

The role of the server process is to receive request messages from client machines and execute them locally as if they were system calls that were initiated on the server machine. When the system call completes, the server returns the requested resource to the client along with any error indication.

The server is a transaction based process, each request is executed to completion before another is begun. Its sole purpose in life is to service requests from remote machines. Multiple server processes can and do exist on any machines that wishes to provide file service. Servers are not associated with any particular client machine or client process.

After receiving a request the task of the server is to masquerade as the requesting process. The request message contains enough information (e.g. uid, gid, ulimit) for the server to appear to the remote machine as the client process. Being a kernel process, the server can extract data from the request message and store it directly into its own user area.

Before fulfilling the request the server must recreate the environment that the client has own its own machine. This varies depending on whether this request was due to a pathname or file descriptor based system call. For a pathname based system call the pathname is set up from the request message. The pathname in this case is the pathname remaining beyond the remote mount point. The pathname evaluation will proceed from the inode of the advertised resource. For file descriptor based system calls the inode specified by the RFS inode on the client side is used to complete the requested system call. After the environment is setup the server executes the specified system call. When the system call completes a response message containing the requested resource and error status is sent to the client.

In the cases where the client will subsequently access the server inode (e.g., *open()*, *chdiren()*), a communication pointer to that inode is included in the response message to the client. In these cases, to preserve the UNIX semantics, state information reflecting the clients reference is recorded on the server machine. That is, the inode reference count on the server inode is incremented to its prevent premature removal. Consider the case where a client process opens a remote file. If another process attempts to remove this file the inode will remain intact because the reference count remains high. If the reference count were not incremented when the client opened the file the inode would be prematurely removed causing the clients file operations to malfunction, violating the UNIX file system semantics. In addition to inode reference counts, file/record locks, reader/writer counts for named pipes, are also recorded on the server inode.

Since client state information is "recorded" on the server, a recovery mechanism for "erasing" this state in the event of a client crash was designed. The details of the recovery mechanism are described below.

Recovery

The main purpose of the recovery mechanism is to restore state on the server machine in the event of a client machine crash, and to cleanup a client machine in the event of a server crash. Recovery is based on the existence of a virtual circuit between the two machines. The underlying transport provider signals the recovery mechanism when a virtual circuit breaks, indicating that the other machine crashed. RFS does not distinguish between a network failure and a crashed machine. The recovery procedure for clients and servers is different.

On the client side the recovery process wakes up any client process that is waiting for a response from the crashed server and marks RFS inodes indicating that the link went down, so that subsequent operations on these inodes will fail. It is important to note that a client process awakened by recovery return an ENOLINK error to the user indicating the server machine crash. The client recovery mechanism also sends a message to a user level daemon which initiates a user level recovery procedure.

On the server side the recovery process "undoes" any state that the crashed client has recorded on the server. This is done by maintaining a per client record for each accessed inode (Figure 9). This record contains the number of references the client has to the inode. If the client machine crashes the server then knows how much to decrement the inode reference counts. These records not only contain inode reference count information, they also contain reader/writer counts for named pipes, so reader and writer synchronization can be restored in the event of a client machine crash. The server recovery mechanism also removes any file/record locks that a crashed client machine has placed on any of its files, to prevent other process from blocking indefinitely. This is accomplished by recording the system identification of the client machine in the file/record lock structure when the lock is set. In the event that a client machine crashes all file/record lock structures with the system identification of the crashed machine are removed.

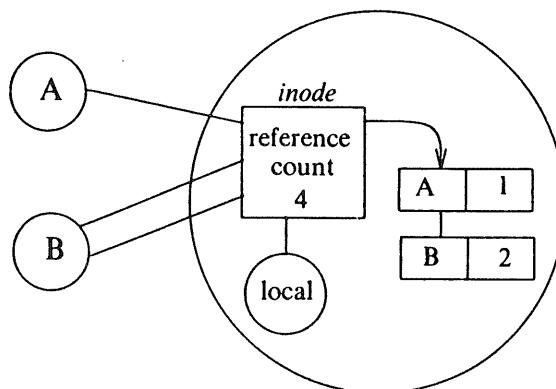


Figure 9. Server Recovery

5. Interesting Issues

In the course of the development several interesting issues were encountered. Some of these issues concerned special devices, time, and *stat()*. These issues are described below.

Special Devices

There were special considerations made for supporting special devices. Since UNIX treats special devices and named pipes as part of the file system most of the job was done. However, three problems had to be solved. First, slow devices and named pipes could consume all available server processes, making the server machine unusable. Second, when a client process sleeps waiting for I/O, a remote signal mechanism must be available to allow the client process to "break out" of the sleep. Third, the remote data movement between the server and the client must be transparent to the device driver.

The first problem was solved by using a dynamic pool of kernel server processes. If all free servers are busy and a client request is received, RFS creates a new server process to handle the request. The size of this server pool is limited by minimum and maximum tunable parameters. In the event

that the pool reaches maximum size the last kernel server is prevented from sleeping, to avoid a deadlock situation.

To solve the second problem the signal mechanism was extended to allow remote signaling. The first step was being able to uniquely identify a process within a distributed environment. This is accomplished by using a system identification (sysid) in association with a process identification (pid). The sysid uniquely identifies a particular machine, while the pid uniquely identifies the process within the machine. The remote signal mechanism is described below.

A client process that has sent a interruptable system call (e.g., *read()*,) request may sleep waiting for a server to complete the I/O operation. If the process receives a signal, a signal request message containing the clients pid and sysid is sent to the server machine. On the server machine, a server process services the signal request by using the sysid and pid to locate the server process sleeping on the I/O operation, and posting the specified signal to that server process (Figure 10).

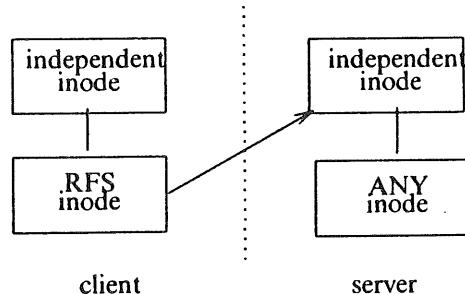


Figure 10. Remote Signal

To isolate the problems of remote data transfer from the device drivers, remoteness detection had to be done at a level below the device driver. The *copyin/copyout* routines are a standard interface used by device drivers to transfer data between kernel address space and user address space. By doing remoteness detection at this level, remote data movement would be transparent to the device driver.

The *copyin* and *copyout* routines have been modified to check if a server process is attempting the data movement. This is done by checking a flag in the processes process table entry, a special flag has been reserved in the process table to differentiate between server processes and regular processes. If it is a server process, control is passed to the RFS module which transfers the requested data to or from the client machine. Upon completion of the data transfer control is returned to the initiating routine.

In addition to data movement the *ioctl()* system call is greatly simplified by the *copyin/copyout* mechanism. A remote *ioctl* system call is passed to a device driver through a server process servicing a *ioctl* request. In response to the *ioctl*, the device driver may read or write data to a supplied address using the same *copyin/copyout* interface. As before the remote data transfer will be transparent to the device driver, making the remote *ioctl* implementation quite easy.

Time Skew

Time is a problem when the current time is different between client and server machines, causing an inconsistent view of file age. This may cause time sensitive commands such as *make*, *news*, *mail* to break.

This time skew problem was solved using a time delta approach. Upon establishing a connection between a client and a server, the time delta between the two machines is calculated and recorded. Time based information sent in response to a client request (e.g. *stat*) would be modified using this delta to compensate for the inconsistency between the two machines. The time delta is recalculated when either machine changes its current notion of time.

stat()

In a distributed environment, where unique devices numbers are not guaranteed, applications using *stat()* to obtain the device and inumber of files to check equality may break. To solve this problem the contents of the device field are modified for remote files. More precisely the upper bit of the device field is used to indicate whether the file is local or remote. The next seven bits are used to record the server machine on which file exists. The last eight bits are used to record the mount table index of the file system in which the file resides. With this scheme the device field will be unique over a network environment, making device/inumber comparisons work.

6. Conclusion

The design of RFS clearly exhibits AT&T's commitment to providing truly transparent file access without compromising the UNIX file system semantics. By maintaining client state information on the server machine to assure data integrity and consistency RFS can be used by existing applications with no fear of malfunction. Allowing access to remote special devices allows users to share expensive peripheral devices easily and economically. In all RFS provides a fully distributed file sharing environment.

7. Acknowledgements

There are many people who contributed to the ideas and spirit of the RFS project. The project was supervised by Art Sabsevitz. The prototype system from which much of RFS was based, was done by Dave Arnovitz and Jeff Langer. We would like to thank Tom Houghton, Steve Buroff, Gil McGrath, Laurence Brown, Maury Bach, Her-daw Che, Mike Padovano, Anil Shivalingiah, Al MacPherson, and Ron Gomes for their help in designing and debugging the system. Also, we would like to thank Peter Weinberger of the UNIX Research Laboratory of AT&T Bell Laboratories for his help during the early stages of the project.

REFERENCES

1. D. M. Ritchie, "A Stream Input-Output System," *AT&T Bell Laboratories Technical Journal* 63(8) (October 1984).
2. D. E. Kevorkian, "System V Interface Definition" *Spring 1985 Issue 1*
3. D. J. Olander, et. al., "A Framework for Networking in System V" *USENIX Conference Proceedings*, Atlanta, Georgia (June 1986).
4. SUN Microsystems, "External Data Representation Protocol Specification" (April 15, 1985).

News from Finland

UNIX† and the polar bears

*Johan Helsingius
(julf@penet)*

Disclaimer:

This is not an official statement from FUUG. This report is based on my completely and purely personal opinions, and in any case, I don't know what I'm talking about!

Oh, well, I have to try to avoid overdoing all those iceberg jokes, polar bear jokes, sauna jokes etc.

As you can see, I am trying to write this thing in some kind of pseudo-English, instead of writing in Finnish. I do this partly to make the story a bit more readable for the rest of you, but mostly out of pity for the poor devils who have to typeset this text. But, well, I just can't resist putting in a couple of really beautiful phrases in Finnish:

"Tulitko konekirjoittajattarettasi?" - "Did you arrive without your (female) typist?"

Which leads to: "Haayoiae" - "Wedding night intentions", the a's and the o's should have dots on top.

And one really beautiful one for the hyphenation algorithm: "kaivosaukko".
"kaivo-saukko" - "Otter living in a well",
"kaivos-aukko" - "Mine entrance".

So much for comparative linguistics, and back to the ongoing story of UNIX in Finland ("So close to Russia, so far from Japan.."). Here you have some boring facts about FUUG, the Finnish UNIX Users Group, or "Suomen UNIX-kayttajien Yhdistys", if you prefer, with dots floating overhead:

The number of members is 113, consisting of 35 institutional members, 75 individual members, and 3 student members. Of the 35 institutional members, 5 are academic and 30 have commercial tendencies.

There are 32 sites on the Finnish UUCP network. The backbone machine is called "penet", and has finally been interconnected to the rest of the European backbones by X.25.

The board of FUUG is composed of the following esteemed members:

Chairman	Johan Helsingius	Oy Penetron Ab	julf@penet
Vice-chairman	Timo Hirvonen	Oy Olivetti Ab	thi@olisf0
Secretary	Ari Kyhala	Oy Mercantile Ab	ajk@me-ncr
Treasurer	Lauri Lounasheimo	Nokia Data	

Members:

Par Andler	Hewlett-Packard Oy	pivi@hpuhela
Mika Arkiomaa	Systecon Oy	
Hannu-Matti Jarvinen	Tampere Univ. of Techn.	hmj@tut
Jari Mantsinen	ETLA	jma@etlahp

Now that the boring facts have been dealt with, we come to the truly interesting question, "In what way is the Finnish UNIX marketplace different from all the other European countries?". Good question! Next question, please!

Ok, ok, I'll try.

† UNIX on AT&T Bellin Laboratorioiden Yhdysvalloissa ja muissa maissa rekisteroima tavaramerkki.

- We don't belong to the EEC. No ESPRIT, haha..
- We Finns are naturally suspicious about new ideas from 'outside'. You see, if 'they' are right, then we are wrong, and as we can't be wrong, 'they' must be!
- The Finnish industry is strongly biased towards 'heavy metal', e.g. shipbuilding, paper mills etc. This makes real-time process control a big hit. UNIX isn't.
- Another factor is that a significant part of our export trade goes eastwards. As Sheriff Reagan and his crew have decided that UNIX is a highly secret weapon, we must use proprietary operating systems or Yugoslavian V7 clones instead. Yacc!
- Finally, Sweden has gone into UNIX in a big way. Now, we can't follow every whim the Swedes do come up with, can we? Of course not, so no UNIX here, please! (By the way, do you know how to sink a Swedish submarine? Just knock on the door! Ha, ha, ha!) [Editorial Note: Did you hear that the Finnish Navy lost its submarine fleet last spring, when they installed the screen windows for summer?]
- It's cold and dark here! Anybody in need of UNIX gurus with experience handling reindeers?

Get the picture?

Most of the universities have at least some UNIX machines, and are happily hacking away on their 4.2 kernels. The rest of the Finnish UNIX world is busy porting their accounting packages written in BASIC or COBOL onto all kinds of existing 68000 System III/System V boxes.

We do have a couple of domestic computer manufacturers, but they aren't currently making any UNIX machines. According to rumours, we seem to have one (1) commercial UNIX source licensee.

Well, the picture I've given you might be a bit pessimistic, but I've just gotten back from the Florence conference to find my house glowing in the dark because of radioactive fallout, and the EUUGN deadline is coming up, so I feel worthy of my latest nickname, "The Flaming Finn".

If you don't believe this report, why don't you come up and see for yourself. The best opportunity will be the EUUG Spring Conference 1987, to be held on a ferryboat between Finland and Sweden. No kidding!

Cheers,

Julf

{seismo,decvax,philabs,garfield,okstate}!mcvax!penet!julf

Johan Helsingius ("julf")
 Penetron
 PL 21
 SF-02171 Espoo
 FINLAND ("..so near to Russia, so far from Japan..")

"What did you say? No polar bears?"

DKUUG since Paris 1985

Keld Jørn Simonsen

keld@diku.uucp

DKUUG

Centre for Applied Datalogy

University of Copenhagen

Changes for DKUUG since April 1985

I will here outline the news and changes which has happened to DKUUG since we last reported at an EUUG conference — the Spring 1985 conference in Paris. The main activities concerns organisational items, meetings, publications and the network.

DKUUG organization

DKUUG — "Dansk UNIX-system Bruger Gruppe" as it is called in Danish — has been around since 18 Nov 1983. Today (April 1986) it has a membership of 115 organisations paying DKK 900 a year and 10 individuals paying DKK 300 a year. This totals to 125 members today, which compared to the April 1985 membership of 92 gives a growth rate of about 30 %. The 125 members are distributed among 75 companies (both hardware and software), 20 universities and other academic institutions, and 30 ordinary users. This indicates that the commercial and university markets for memberships are quite satiated, but with an estimated 1000 UNIX machines sold in Denmark (per annum?) it leaves plenty of room for new user memberships.

We had our statutes changed on our general assembly meeting 85-11-28, so that the installation and associated membership classes merged to one class, called organisational members.

Meetings

We have had four member meetings since Paris, with 25 to 90 people attending each one. Most discussions are held in the Danish language. There have been no exhibitions during this time. The items discussed have included: big UNIX system users, public domain software, 4.2 BSD systems administration, a market survey from IDC, databases, help systems, administrative systems, the network, and UNIX courses. In addition, there were four machine presentations; the NCR Tower, ICL Comet-32, RC39 and the Supermax were all shown to the membership.

DKUUG also hosted the EUUG Copenhagen Autumn 1985 Conference, which was a huge event for us, with 350 participants and 4 days of meetings and exhibitions. This conference has already been covered in detail in the EUUG newsletter.

As a new feature, we have introduced a suppliers board within DKUUG, in which we discuss demonstrations, make catalogues of hardware, review software & services available in Denmark, and discuss how to attract more members to the DKUUG.

The DKUUG board has 6 members: chair, treasurer, editor, network manager, suppliers coordinator and EUUG governing board member. We hold about 10 board meetings a year.

Publications

Our newsletter is called "DKUUG-nyt" and we have made 5 issues since April, 1985. There have been various articles on meetings, the network, news, rumours and an important survey of courses.

A large amount of work has gone towards developing the book "UNIX-bogen", which is a Danish translation and update of "UNIX - The Book", by M. Banahan & A. Rutter. "UNIX-bogen" is updated to correspond to UNIX System V and 4.2 BSD, and also covers the special concerns of handling Danish language within this American developed operating system.

Also in the area of language, we have made our own description of UUCP in Danish, and we are distributing other Danish articles on UNIX, C, and other topics. Finally, we are revising the 2 year old DKUUG - UNIX folder — a lot has happened these two years.

We have also made several good agreements for our members for outside publications, including a 40% discount on subscriptions to 'PC world' magazine, 25% off for 'UNIGRAM/X' and also 25% off for the book 'UNIX-bogen'.

The UNIX network in Denmark April 86

We have recently registered EUnet and DKnet as trade marks, to prevent others from using the names. Also we have recently installed the rerouter and begun a campaign to get more 'netters'. Our future plans are to get mail accounting working, to use X.25 for outgoing international connections, and to offer a 2400 bps V.22 bis service.

The current number of members using the net services and the costs thereof are:

Service	Apr 86	Apr 85	change	cost/year
Mail	24	21	+ 3	DKK 500
News	4	4	+ 0	DKK 8000

While we have not experienced much growth here, we are expecting a boom after our net campaign.

Abstracts from the Florence Technical Programme

Malcolm Agnew **DB + + Database Management System**

Mail: robert@hslrswi
Phone: + 69 597 029798

Concept ASA GmbH
Wolfgangstrasse 6
D-6000 Frankfurt am Mainl
WEST GERMANY

The DB + + family of programs together comprise an efficient, flexible and reliable relational database management system for use with UNIX.

This paper discusses how the DB + + programs have been fully integrated into the UNIX framework. It then goes on to explain the choice of query language in addition to some of the unusual implementation details.

Charles Bigelow **Florentine Inventors Of Modern Alphabets**

Mail: ukc!cab@su-ai.arpa
Phone: + 1 415 788 8973

Bigelow and Holmes
15 Vandewater Street
San Francisco
CA 94133
USA

Bitmap screen displays and laser printers have enriched the appearance of computer literacy. Procrustean limitations of mono-case and mono-space that formerly degraded computer-produced text have been abolished. We now can enjoy the luxury of reading and printing text in lower-case as well as in capitals, in italic and bold styles as well as in roman, in proportionally-spaced fonts of different sizes as well as in monospaced fonts of a single size, in justified as well as in ragged-right columns.

Massimo Bologna
The Portable Common Tool Environment Project

Phone: +39 50 500211
Mail: 3bicoa!flor@iconet.uucp

The Portable Common Tool Environment (PCTE) project is carried out as part of the ESPRIT programme of the Commission of European Community.

The project aims at the definition and the implementation of a common framework, within which various software tools can be developed and integrated in order to provide a complete environment for software engineering.

The two main goals of the project are the portability across a wide range of machines and the compatibility for assuring a smooth transition from existing software development practices.

The technical approach to portability is discussed in this paper: UNIX is the means by which PCTE will be widely available; the PCTE basic mechanisms are built on top of UNIX: in fact PCTE can be seen as an extension to UNIX in the area of distribution, friendly user interfaces and database for software engineering.

This last aspect is described in this paper: functional as well as implementation aspects will be dealt with.

Finally, tools developed for the PCTE database are described and discussed.

C. Brisbois
SIGMINI Information Management System

Union Miniere SA
Division Information
Avenue Louise 54, BTE 10
B-1050 Brussels
BELGIUM

Sigmini is designed to process heterogeneous information, including quantities, which have complex interrelationships. It is related to both database systems and classical documentation retrieval systems. In either case, Sigmini is designed to avoid the necessity to declare in advance the data or relationships.

Antonio Buongiorno
Office Data Base Services in a UNIX Architecture

Phone: +39 125 52 15 92

Olivetti DSRI/DPS
IVREA
Italy

Antonio Buongiorno
Franco Calvo
Bruno Pepino

Classification, filing and retrieval are important but time-expensive activities in an office environment. Attempts to reduce times, and thus costs, involved in these processes and to increase the effectiveness of an retrieval system are rapidly gaining importance for a better information management in an office. The paper outlines a solution for filing and retrieving "objects" in an architecture based on UNIX servers and networks of personal computers. The focus is mainly on the data model that supports the Office Data Base and allows a very fast retrieval of structured and non structured information.

In the first section is described a general architecture, in the second the functionalities and the data model and in the third the prototype package AMEDEUS.

Brian Collins
The Design Of A Syntax-directed Text Editor

Mail: ukc!bco@ist
Phone: +44 1 581 8155

Imperial Software Technology
60 Albert Court
Prince Consort Road
London
SW7 2BH

The editing and user-interface system described in this paper presents many different guises to the user. Among these are:

- 1) A general-purpose, multi-file, multi-windowing screen editor for ASCII text files and simple terminals.
- 2) A forms system for constrained data entry and presentation.
- 3) An interactive windowing front-end to application programs.
- 4) A syntax-directed editor.

This paper concentrates on the latter.

In the design of such an editor, the choice of facilities offered to the user is often more difficult than the actual implementation. In this editor, the decision taken was to make the interface appear to the user as close as possible to a standard text editor, hence the more descriptive title of 'syntax-directed text editor'.

Normal text editing operations can be seen to fall into two classes: those which only affect one line of text, and those which affect a number of lines. The user is allowed to edit freely any text within a line, using text-editing operations. On leaving the line, the edited text is re-parsed and errors corrected or reported. Editing operations which insert, delete, move or copy lines must explicitly maintain syntactic correctness across line boundaries. For example, a line containing 'END' could be automatically added when a 'BEGIN' is detected.

The overall effect is to provide a text editor in which it is impossible to enter a syntactically incorrect program.

The editor is language-independent (indeed, it may handle many different languages simultaneously). A language may be supported if it conforms to a few obvious restrictions (similar to those of lex & yacc). The syntax of the language is described in an extended, annotated BNF, which also specifies the layout rules for line breaks, spaces and indentation.

The paper provides a description of the editor, both from the user's view and from the language implementor's. It describes how the syntactic structure is maintained in what is essentially a text editor, and the techniques used for the fast re-parsing of edited text. Finally, the paper assesses the applicability of syntax-directed editing to various languages, its use in a programming environment, and a specific analysis of the problems in the syntax of the language C.

Pete Delaney

A Guided Tour of OSI based NFS

Mail: ukc!ecrcvax!pete@unido.uucp
Phone: +49 89 92699 138

Rockey Mountain UNIX Cons
ECRC
Arabellastrasse 17
D-8000
Munchen 81
WEST GERMANY

A guided tour of the ESPRIT OSI implementation on 4.2BSD will be presented. An example of mounting a filesystem via Sun's NFS using OSI protocols over X25 will be demonstrated with:

- A Kernel trace showing major events
- A diagram of chronological events, with references to the kernel trace.
- Topological organization of network structures.

Due to the lengthy nature of the trace, copies will be distributed to the audience. Implementation details and divergencies from the US NBS and General Motors MAP project will be discussed.

The merits and pitfalls of the OSI protocols will be presented along with recommendations for further work.

Winfried Dulz

System Management for a Distributed UNIX Environment

Mail: ukc!fauern!faui70!dulz@unido.uucp
Phone: +49 9131 857929

IMMD 7
Martensstrasse 3
8520 Erlangen
WEST GERMANY

In analogy to system management for central server configurations, distributed systems must also provide utilities for user-handling, system- accounting and resource-accessing. We show for the UNIX network operating system Newcastle Connection how transaction-oriented protocols based on communicating client/server processes can solve this class of problems. To that end all local user-files /etc/passwd are concatenated to a system-wide database that also contains information about login hosts and UNIX systems that can be reached by means of the Newcastle Connection. The only process that may update this database is a reliable and redundant system process that guarantees the necessary data consistency.

William Fraser-Campbell

Implementing the NFS on System V.2

Mail: ukc!inset!bill
Phone: +44 1 482 2525

The Instruction Set Ltd
152-156 Kentish Town Road
London
NW1 9QB

System V has been enhanced to support multiple file system types using a common interface within the kernel.

Using the Sun Network File System architecture, our implementation supports System V files on local disks, and acts as both client and server for network file systems using published NFS protocols.

This paper will present details of the implementation.

M Guarducci
Fiore Project: Wide Band Metropolitan Area Network

Electronics Department
Florence University
Florence
ITALY

The design and implementation study discussed in this paper outlines first problems and solutions of the design work, followed by implementation strategies of services supplied to final users: file transfer, messages, electronic mail.

The realisation foresees use of host computers running the UNIX operating system connected on the wide band based MAN of the Fiore project in Florence based on the Localnet 20 protocol by Sytek, on which Ethernet LAN's and stand-alone computers may be connected.

Mike Hawley
Developments at Lucasfilm

Mail: ukc!mike@dagobah
Phone: +1 415 485 5000

PO Box CS 8180
San Rafael
CA 94912
USA

Mike will discuss UNIX & computers at Lucasfilm. The excitement comes from combining information technology with the richest possible kinds of communication media. Examples range from the high-end graphics and audio work done there (with the PIXAR and ASP systems) to earthier projects involving large databases of sound effects, books, poetry, etc, and of course, music.

Most of the work exemplifies UNIX applications and systems development with an artistic bent.

Robert Heath

Adding Commercial Data Communications to UNIX

Phone: +1 513 445 6583

NCR Corp.
Columbia, South Carolina

Tlx: 851295467CZ8123Z 295467

As the UNIX operating system becomes more widespread in small business systems, the need to add commercial data communications becomes important. This paper discusses how NCR in its UNIX-based supermicrocomputer, the Tower, has supplemented the basic data communications tools provided by AT&T with both industry-standard and internationally standard protocols. The resulting product provides interconnectability in three general areas: asynchronous, synchronous, and local area networking. The paper illustrates which protocols are important for a general-purpose small business system.

The well-known Call UNIX (CU) and UUCP are standard Tower utilities for asynchronous networking. A high-performance, asynchronous adapter, which moves the tty driver off the main processor for both networking and workstation control is described.

UNIX System V is deficient in synchronous protocols. Described is an intelligent, synchronous adapter which supports multiple, block-oriented protocols such as SDLC, HDLC, and Bisync. Data link control drivers were added to the kernel to complement standard networking packages such as SNA, X.25, and 2780/3780 Bisync, and 3270 Bisync. The paper describes how 3270 screen emulations reuse UNIX concepts such as termcap and remote job entry emulations reuse the printer spooler.

Tower local area networking (Towernet) is provided through a programmable Ethernet adapter which offloads time-critical operations. Towernet services described include electronic mail, file transfer, virtual terminal, PC interconnection, and wide-area networking. Another lan-based option is the distributed resource system which not only implements a distributed file system but also provides transparent access of remote devices.

This paper illustrates how modern, layered protocols are distributed within UNIX. It details particular problems in intertask communications and multiplexing separate data streams. Solutions to these problems are presented in application software, kernel software, firmware, and hardware. Strategies for network management, diagnostics, and maintenance are offered.

Bill Joy

UNIX Workstations: The Next Four Years

Mail: ukc!inset!sunuk!sun!wnj

Phone: +1 415 960 1300

Sun Microsystems Inc.
2550 Garcia Avenue
Mountain View
CA 94043
USA

At the EUUG conference in Paris 1982 Bill predicted the future of UNIX workstations and the technology associated with them for the three years which were to follow. That time has elapsed and therefore he will update that talk with further insights into the developments in the next few years.

The talk will not only cover UNIX workstations, but developments in the technology applicable, regardless of operating system.

Tom Killian
Computer Music under UNIX Eighth Edition

Mail: ukc!tom@ikeya
Phone: + 1 201 582 3000

AT&T Bell Laboratories
Murray Hill
New Jersey
USA

We describe an evolving computer music system which draws upon many of the novel facilities of the 8th edition as well as the standard repertoire of Unix tools. The Teletype 5620 bitmap display serves both as the user's terminal and real-time controller. The mux window system is used to download a MIDI interface driver which services other windows (by direct code sharing) and host processes (which write on the driver's control stream). We presently have two MIDI-compatible instruments, a Yamaha DX7 and TX816.

Window programs include a piano-roll style score facility and a virtual keyboard. Host programs include a music compiler, "m," which converts an ASCII score notation into MIDI events; it is based on lex and yacc, making it very easy to develop in response to user needs. There is also a variety of filters which perform simple transformations (e.g., time and pitch translation) on MIDI files. The latter are ASCII, so that, for example, output from the DX7 keyboard can be translated into "m" notation with an awk script, and other Unix text filters (especially sed and sort) and "c" programs are useful as well.

We will show (and play) examples of pieces written in "m," "c," and the Bourne shell, and discuss future plans which center around 12-tone serial composition.

Tom Killian began his career as an experimental high-energy physicist at CERN and Brookhaven National Laboratory. Frustrated in his attempts to persuade his colleagues of the evils of JCL, he eventually found his way to Bell Labs, where he has been a member of the Computing Science Research Center since 1982.

S Mecenate
The IBM 6150 Executive AIX

Phone: + 44 1 995 1441

IBM UK Ltd
Chiswick
London W4

In January 1986 IBM has announced the IBM 6150 micro computer with the Advanced Interactive Executive (AIX) operating system.

As the base for AIX IBM chose AT&T's UNIX System V because it provides considerable functional power to the individual user, multi-user capabilities, is open-ended, and has a large user and application base.

However in choosing UNIX IBM recognised the need to make significant extensions and enhancements to meet the needs of our expected customers and their applications.

The presentation will discuss some of these major modifications and additions.

Marco Mercinelli

SNAP: Restarting a 4.2 BSD process from a snapshot

Mail: ukc!cse!marco@i2unix.uucp

Sezione Metodologie Software
Divisione Informatica
Centro Studi e Laboratori Telecomunicazioni
10148 Torino
ITALY

SNAP is an extension to UNIX4.2BSD for taking a snapshot of a running process and restarting its execution at a later time.

A snapshot is composed of a process core image and information about its execution environment (opened files, devices, tty settings, etc.). It is not necessary to kill the process for taking the snapshot.

A process can be restarted at any time, even after a system crash. Its execution continue at the point the snapshot was taken in a "quasi" transparent fashion. All the process resources should be available and are set to a suitable state.

The snapshot facility can be used as a building block for several higher level mechanisms such as crash recovery, debugging, backtracking and process migration.

The current implementation of SNAP can only restart a single process with no IPC connections. Further work is needed in order to extend SNAP for managing groups of related processes and connections in a distributed environment.

Roberto Novarese

An Office Automation Solution with UNIX/MS-DOS

Phone: + 39 125 52 15 92

Olivetti DSRI/DPS
Ivrea
ITALY

In this paper a set of Office Automation requirements of the Economic European Community are presented. A solution that has been designed and prototyped by Olivetti is then discussed. The proposal is based on the integration of UNIX mini and MS-DOS personal computers on a local area network. A set of integrated Office Productivity Tools on the workstations provide the support to professional and secretarial activities.

Personal Computers Support Services provide the sharing of resources (file server, print server and communication server functionalities). An X.400 Electronic Mail and an Archiving System are the Office Cooperation Services designed for the integration of this solution in a Multivendor Architecture.

Philip Peake
Implementing UNIX standards

Mail: ukc!philip@axis
Phone: + 33 1 4603 3775

Axis Digital
135 Rue D'Aguesseau
92100 Boulogne
FRANCE

As UNIX becomes more firmly established in the commercial computing world there is much pressure, and resulting action for standardisation. For example, the SVID and X/OPEN publications.

This presentation looks at some of the problems encountered during the development, and subsequent porting of applications which either run on, or communicate with UNIX systems. Some attention is also paid to the existing standards; as found in practice, and as proposed in the above documents.

Dave Presotto
Matchmaker: The Eighth Edition Connection Server

Mail: ukc!presotto@research
Phone: + 1 201 582 5213

AT&T Bell Laboratories
Murray Hill
New Jersey 07974
USA

Matchmaker is a connection service for Eighth Edition UNIX. Using Matchmaker, processes can connect to processes on the same system or across a variety of networks. Unlike other solutions to this problem, such as 4.2 BSD's sockets, ours separates network protocols and communication properties to such an extent that application programs using it need not be cognizant of the network or network protocol on which the connection is built.

Matchmaker is based on Dennis Ritchie's Streams, a mechanism for providing two way byte streams between processes and devices or between processes and other processes. The unique properties of Streams make Matchmaker possible. Using Streams we can perform functions such as circuit setup, circuit shutdown, and data stream processing in the kernel, in processes, or even in separate processors as the situation dictates.

John Richards
Software for a Graphics Terminal in C

Mail: ucl-cs!richards@uk.ac.bristol.qvc
Phone: + 44 272 303030

University of Bristol Computer Centre
University Walk
Bristol
BS8 1TW

This paper describes the development of software for incorporation in a new intelligent raster graphics terminal. The terminal provides support for windows and graphics segments. The software was written in C and developed and tested on a UNIX™ system before being placed in ROM in the terminal. The paper shows how considerable use was made of structures and the storage allocation functions to provide a generalised segment storage scheme. Examples are given of the way language constructs were used to obtain fast, but portable, code. The finished software was ported to the terminal with hardly any modifications.

Andy Rifkin
RFS in System V.3

Mail: ukc!uel!attunix!sfjec!apr
Phone: +1 201 522 6283

AT&T Information Systems
190 River Road
Summit
NJ 07901
USA

Andy is a principal developer of AT&T's distributed UNIX file system known as RFS. He joined AT&T after receiving his masters degree in computer science from Cornell University.

The AT&T distributed file system known as RFS, provides users with transparent access to remote filesystems. The goal behind RFS is to offer the complete functionality of the UNIX filesystem (i.e., special devices, record locking, named pipes) without compromising the UNIX filesystem semantics. That is, programs which run in a single machine environment will run in an RFS environment with no change.

The implementation of RFS is done at the kernel level, using the standard UNIX mount command to access remote resources. RFS was designed to be both protocol and media independent using the Streams architecture available in UNIX System V Release 3.

This talk will describe the RFS architecture from both a communication and kernel perspective. In addition a comparison between RFS and other available distributed file systems will be made in order to highlight their differences.

Russel Sandberg
Design and Implementation of NFS

Mail: ukc!inset!sunuk!sun!phoenix!rusty
Phone: +1 415 960 1300

Sun Microsystems Inc
2550 Garcia Avenue
Mountain View
CA 94043
USA

The Sun NFS provides transparent, remote access to filesystems. Unlike other remote filesystems available for UNIX, NFS is designed to be easily portable to other operating systems and architectures.

This paper describes the design and implementation of NFS along with some experience of porting it to other systems.

David Tilbrook
Managing a Large Software Distribution

Mail: ukc!dt@ist
Phone: +44 1 581 8155

Imperial Software Technology
60 Albert Court
Prince Consort Road
London
SW7 2BH

One of the major problems at IST is the management of more than eight mega-byte of software and related data for IST's own machines and those of our clients. For obvious reasons it is desirable to centralize the management of the source in a single location and to extract the distributions as required. This presents a variety of problems, principally with respect to the variations in the target systems. This paper discusses the solutions developed to overcome the differences between operating systems, interdependncies within the source, the variations in the available software tools.

Peter Weinberger
The Eighth Edition Remote Filesystem

Mail: ukc!pjw@seki
Phone: + 1 201 582 3000 ex: 7214

AT&T Bell Laboratories
Murray Hill
NJ
USA

Peter is famous for at least two pieces of work. Firstly, he is the W in AWK (Aho, Weinberger and Kernighan), that useful pattern matching and scanning language we all use daily. His second well known work is in the area of distributed filesystems, in particular he was responsible for the initial design and implementation of the Edition VIII remote filesystem.

In Florence, Peter will explain the motivation behind this work on the remote filesystem and study the model of the world it assumes.

Lauren Weinstein
Project Stargate

Mail: ukc!lauren@vortex
Phone: + 1 213 645 7200

Computer/Telecommunications Consultant
PO BOX 2284
Culver City
CA 90231
USA

This paper and talk will review the current status of the ongoing "Stargate" project, which is experimenting with the transmission of "netnews"-type materials over the satellite vertical broadcasting interval of television "Superstation" WTBS, a very widely available basic cable television service in the United States. The techniques used allow the Stargate data to exist in parallel with the standard video and audio of the television operation. Satellite-delivered WTBS is currently available to over 33 million cable subscribers (households and businesses) throughout the United States, and is also received directly by privately owned satellite earthstations. This paper discusses both the technical and non-technical (i.e. organizational, content, policy, etc.) aspects of the project.

The Florence Competition

Peter Collinson

Secretary — EUUG

We have had many silly competitions at many conferences but the one in Florence attracted the most entries to date. It was probably the promise of a bottle of Champagne which focussed people's minds — but it was a good basic idea and was also some fun.

The problem was to invent a new error message and a mnemonic name for the message. If you are unfamiliar with the introduction to Section II of most UNIX manuals, then you should know that error messages which are returned by the operating system kernel have a name starting with the letter 'E', and a value which is usually irrelevant. A routine **perror** translates these numbers back into visible strings which many programs then print out as an error message to you, the humble (and often mystified) user.

So here are (nearly all) the entries, we have some names but not all, so all names are suppressed to prevent unjust accusations of timewasting from the people paying for the trip.

E3PO	Robot dumped
ENOINSTRUCTIONSET	Microcode failure — <i>hard to get this back to the program</i>
EBGB	Too early (before Great British Time)
EOK	Doesn't conform to standards
EFREEZE	Too cold to start
EVAPOWARE	Feature not implemented, please recode
ENOSTD	Standard not defined for this feature
ECANTDOIT	Your operation cannot be carried out
EBUGFOUND	(char *) NULL
EBYGUM	Attempt to walk and use System V semaphores, simultaneously, concurrently, asynchronously or at the same time
ESVID	Attempt to use a standard system call
EIEIO	Farmyard mail
EEXPORT	Feature cannot be exported from the US
ELODPOD	LOGic error Delayed until PrOgrammer Dead
EYORE	Fundamental error
ENOP	Operating system not present
EZPZ	No trouble at all (<i>helps if you read in this in American</i>)
ERIP	Process died
EEC	Error Executing C
EAHUM	I am thinking about that one!
EHOTTUB	Feature only works in California
EBYGUM	'trouble at mill' general purpose diagnostic proposed by Lancashire User Group as part of the UNIX dialectisation effort
EEZY	Task not sufficiently challenging
EGONRONAY	Lunch break forced
E+ +	Error producing conference papers
ELLANFAIRPWLLGWYNGYLLGOGERYCHWYRNDROBWLLLANTYSYLIOGOGOGOCH	String too long †

† A place in North Wales, meaning "St. Mary's (Church) by the White Aspen over the Whirlpool and St. Tysilio's (Church) by the red cave".

ELLANFAIRPG	System V version of the above‡
EDTNC	No dt, No Comment
ENOTONUNIX	Not a UNIX system
ET	Phone home
EI2U	Error in Italian User Group
ENOVAX	Sorry, no VAX
ENOFOOT	Footnote missing
EEC	Warning 8-bit character in use
EMSTD	Too many standards
ENOCOFFEE	Queue too long
ESEE	Inspect code carefully
E42	Answer found
ELT36	too few bits
ENOLICENCE	Bootleg kernel
ENOLICENSE	American Bootleg kernel
e=mc ²	Sub-atomic integrity violation (distributed garbage collection in progress)
ENOTUNIX	Attempt to do anything sensible on a non-UNIX system
ESQUAWK	Something queer in awk
E134	EEC meaningless standard error
ENOMONEYNOFUN	Insert more coins
EOSDMCC	Operating system drunk methanol and can't C
EMACS	Machine crunching process
EMACS	Over consumption of resources
EPOF	You have forgotten the power switch
ENIH	Wrong remote file access method
ENAMETOOLONG	Recursive operation
F	Too many possible errors
EOF	Oh f*** it anyway
E}	Wrong baud rate
EbHUH	Wrong bit size
EPHUH	Processor not found
ERROTIC	CPU turned on
E)	Line error
E2SMART	Device or controller has a brain the size of the universe; can't be bothered with your stupid little read, anyway
e ^{iπ}	Root port onto negative login hardware
ENOTFOUND	Wrong include file
EIQ0	User error
EIEIO	Error in ethernet interface I/O
ENOALCHOHOL	Failure on stdin
EXPENSIVE	Social dinner cancelled
ENOJOY	Speaker not found
EBCDIC	Extra big character detected in crash
EGLIDER	Pilot dumped
ECHO	Chronic hangover error
ETC	Tilbrook comment error

‡ Llanfair P.G is the local's name for the place.

EHAL	SNA response to NRS/RFS
ETMFST	Too many file system types
EALP	(uucp) only 110bps South of the Alps
ENOBAR	You have a Danish terminal
EYORE	Donkey not found
EDCBA	rev command not found
EEEEEK	/dev/mouse escaped

The Jury had to consume many a gin before making some incoherent decisions about the entries.
Rushing into third place was

EUUG Unknown user group

A close second was

E Too verbose

but the undoubted winner was

ENOTOBACCO Read on an empty pipe



;login:

The USENIX Association Newsletter

Volume 11, Number 5

September/October 1986

CONTENTS

Third Computer Graphics Workshop	3
Contributions to ;login:	3
Cogito, An Expert System to Give Installation Advice for UNIX 4.2BSD	4
<i>A. Terry Bahill and Pat Harris</i>	
BSD UNIX Manuals – The Next Chapter	9
4.3BSD Manual Reproduction Authorization and Order Form	13
Access to UNIX Standards	14
Call for Papers: Human-Computer Interaction	15
Call for Papers: Washington, DC, USENIX Conference	16
Tutorials to be Offered in Washington	18
Future Meetings	18
Summary of Board of Directors' Meeting in Denver, January 14-17	19
Summary of Board of Directors' Meeting in Napa, March 19-21	20
Long Range Planning Committee	22
Financial Statements – 1985	23
Book Review: The UNIX C Shell Field Guide	27
<i>Marc D. Donner</i>	
Atlanta Videotapes Available	28
Publications Available	29
NZUUG Offers Kiwis	29
Local User Groups	30
Ten Years Ago in ;login: (a.k.a. UNIX News)	31

The closing date for submissions for the next issue of ;login: is October 31, 1986



THE PROFESSIONAL AND TECHNICAL UNIX® ASSOCIATION

Cogito, An Expert System to Give Installation Advice for UNIX 4.2BSD

A. Terry Bahill
and
Pat Harris*

Systems and Industrial Engineering
University of Arizona
Tucson, AZ 85721

When you try to use a computer, your first effort inevitably fails. You then recall the sage advice, "If all else fails, read the instructions." So you decide to do this. But where do you start? For the tasks described in this paper the instructions were hundreds of pages long spread over a several manuals. So we wrote an expert system to help the human use this data.

In this paper we will discuss *Cogito*, an expert system that gives installation advice for bringing up the UNIX 4.2BSD operating system on a VAX computer [1]. A detailed reference manual is currently used for installation instructions. Task complexity limits this method. *Cogito* filters the information and presents only relevant advice about the user's computer system. *Cogito* remembers data the user has previously entered and uses this to customize its response. *Cogito*, written in M.1 and running on a personal computer, uses if-then production rules to encode the knowledge. *Cogito*'s knowledge base has two components: classification knowledge and process knowledge. Classification knowledge transforms one kind of knowledge into another. Production rules are appropriate for *Cogito*'s classification knowledge. Process knowledge directs the flow of information and the explanation for that knowledge. *Cogito*'s process knowledge is difficult to encode with production rules because in describing the installation process the expert does not think in terms of rules.

Problem Statement

Installing a computer system is an evolutionary process consisting of a series of operations that transform a computer into a complex system capable of supporting many users and functions. There are two basic types of system installation: system building and device integration. System building is required when the computer hardware is delivered and involves installation of the operating system. Device integration is required when a new device is obtained and must be integrated with the rest of the system. This paper will only discuss system building.

The knowledge needed to do system installation is fact-intensive because hardware and software designers have already made many decisions regarding the system structure. Consequently, system installation dictates a long, interconnected series of steps to get the hardware and software to interact correctly. This process is especially complex with the UNIX operating system since it is designed to be as portable as possible, supporting many different types of computers and devices. For example, 4.2BSD UNIX supports three VAX models, three communication buses, 21 disk drive types, 11 tape drive types and 22 device types. Simple combinatorics yields the number of 45,000 minimal systems (one VAX cpu, one communication bus, one disk drive, one tape drive and one console terminal). Almost no one has a minimal system. However, our Systems and Industrial Engineering Department's VAX-UNIX system is close to minimal: it has one VAX cpu, two communication buses, one disk drive, one tape drive and two other devices. For this system there are approximately a half a million combinations.

*Present address Bell Communications Research, New Jersey.

Established Method

Information regarding system installation of UNIX 4.2BSD is contained in the *UNIX Systems Manager's Manual* [2], and the *UNIX Programmer's Manual Reference Guide* [3]. The *UNIX 4.2BSD Systems Manager's Manual* is an extensive reference document that explains in detail the making and installing of UNIX 4.2BSD. The *Programmer's Manual Reference Guide* provides detailed information about specific devices that may be attached to the computer. Unfortunately, these reference manuals are also forced to serve as the only tutorials available for making and installing the UNIX system. They are not well suited for this task for several reasons.

First, the manuals' structures do not provide a model of the problem for system installers to base their learning on. Additionally, the information is often written at a higher level than inexperienced system installers can understand and explanations are sometimes pages distant from the first example. Finally, the wide variety of possible VAX computers, disks, tape drives, controllers, printers, terminals and other hardware makes it impossible for a linear explanation, such as in a book, to distinguish all the relevant information for a particular user. Consequently, to install the operating system with this method, a system installer must painstakingly seek to understand and extract the information needed for his or her system from the vast amount of details for all possible combinations provided by the manuals.

Example Rules

We transformed this vast amount of information into if-then production rules for our expert system. The following shows examples of these rules.

This rule sets up the correspondence between the DEC disk name and the DEC-bus that it is attached to.

```
dt-hp-a: if disk = 'RM03'  
          or disk = 'RM05'  
          or disk = 'RM80'  
          or disk = 'RP06'  
          or disk = 'RP07'  
          then disk-bus = 'MASSBUS'.
```

This rule establishes the first level correspondence between DEC disk names and their UNIX counterparts.

```
dt-hp-b: if disk = 'RM03'  
          or disk = 'RM05'  
          or disk = 'RM80'  
          or disk = 'RP06'  
          or disk = 'RP07'  
          then standalone-disk-name = hp.
```

These rules establish the UNIX-to-UNIX relationship between the standalone disk names and their controller designations. This demonstrates the nonobvious relationships that occur in UNIX-land that are very similar to the English language tradition of irregular verb forms; some forms fit the pattern, while most do not.

```
dtc-1: if standalone-disk-name = hk  
          then controller = hk.  
  
dtc-2: if standalone-disk-name = ra  
          then controller = uda.  
  
dtc-3: if standalone-disk-name = rx  
          then controller = fx.  
  
dtc-4: if standalone-disk-name = up  
          then controller = sc.
```

These two rules show that non-DEC disks can be attached to either the UNIBUS or the MASSBUS, but the UNIX disk types are decidedly different.

:login:

```
dt-26: if disk = 'AMPEX 330M'  
      and disk-bus = 'UNIBUS'  
      then disk-type = capricorn.  
  
dt-34: if disk = 'AMPEX 300M'  
      and disk-bus = 'MASSBUS'  
      then disk-type = 9300.
```

User Satisfaction

Cogito is a better method for system installation than using the *UNIX Systems Manager's Manual*, in terms of overall user satisfaction, because the amount and relevancy of the information presented is significantly increased. For example, in the disk definition state the file */etc/fstab* must be created. Assume that the boot disk is a 'AMPEX 300M' connected to uba0 at drive 1. This implies that the disk address is 1. Also assume the user's name is 'Pat' and that he is ready to install *fstab*. The instructions given in the manual are generic and therefore the human must relate the general instructions to the specific application. Figure 1 is a copy of a the relevant section of the manual.

4.4.1. Initializing /etc/fstab

Change into the directory */etc* and copy the appropriate file from:

```
fstab.rm03  
fstab.rm05  
fstab.rm80  
fstab.ra60  
fstab.ra80  
fstab.ra81  
fstab.rp06  
fstab.rp07  
fstab.rk07  
fstab.up160m (160Mb up drives)  
fstab.up300m (300Mb up drives)  
fstab.hp400m (400Mb hp drives)  
fstab.up (other up drives)  
fstab.hp (other hp drives)
```

to the file */etc/fstab*, i.e.:

```
# cd /etc  
# cp fstab.xxx fstab
```

This will set up the initial information about the usage of disk partitions, which we see how to update more below.

Figure 1: Installation of *fstab* from [2].

Cogito's instructions to complete the same task are:

```
# cd /etc  
# cp fstab.up300m junk  
# vi junk  
(Edit the file, Pat.)  
    a. Add the line '/dev/up0b::sw::'.  
    b. Give the global substitute command ':g/up0/s//up1/'.  
    c. Save the new contents and quit the editor.)  
# cat junk >> fstab
```

Cogito has remembered that half an hour ago the user said his name was Pat and his disk was an AMPEX 300M and has used this information to make its instructions specific. Cogito's instructions are personalized, relevant to the user's task, clear and complete!

:login:

This example illustrates that the manual's instructions are incomplete, and rely on previous knowledge and implicit knowledge: they are incomplete because the swap partition */dev/up0b*, must be added to the file; they rely on previous knowledge because the user must recall that the UNIX standalone disk name for a 'AMPEX 300M' is 'up'; and they rely on implicit knowledge because the user must somehow know that the disk address of the partitions needs to be changed from '0' to '1'.

To address the issue of whether Cogito is the best system for advising on UNIX system installation, qualifications must be made. As originally designed, Cogito has two user classes: a system builder and a system integrator. It's interesting to note that although the knowledge base is the same for both kinds of users, Cogito is inadequate for the system integrator user. All the appropriate information is given to the user but the instrument on which it is displayed is wrong. Transmission of the advice via a personal computer seems inconvenient when the VAX computer is running. Ideally, Cogito should have the capability to run on the VAX computer once it is up and running. This includes the ability to transfer information "learned" by Cogito in the system building stage into a database where it can be retrieved when configuring a new device. Unfortunately, this cannot be done because M.1 only runs on a personal computer.

Information Flow

Information flow is concerned with the data transfer from the user into Cogito and the corresponding transfer of advice from Cogito to the user. The information flow depends on the knowledge base structure. Since Cogito is based on the backward chaining inference engine M.1, both direction flows (into and out of Cogito) depend on a goal-oriented knowledge base structure. Ideally the knowledge base is a true reflection of how the expert thinks about the problem. In Cogito, the knowledge base was originally constructed without reference to a particular inference engine. For instance, an original rule near the end of the configuration state was:

```
if config done then
    # config NAME
    # cd .../NAME
    # make depend
    # make vmunix
```

As implemented in Cogito the control method of the inference engine imposed a backward chaining thought process to occur. Hence, using M.1, the implemented version of the same rule is:

```
if display(' # config NAME ')
and display(' # cd .../NAME')
and display(' # make depend ')
and display(' # make vmunix ')
then config is done.
```

Looking at Cogito from the end user's viewpoint, the questions asked and the advice presented appear to be given in a logical, relevant and concise manner. However, the implemented knowledge base structure of Cogito suffers from an unnatural viewpoint, forced by the control method used in the inference engine.

Choice of Expert System Shell

We had two conveniently available expert systems shells to choose from: M.1, primarily a back chainer, and OPS5, primarily a forward chainer. Although the problem domain seems to be data driven, which would suggest a forward chainer, we found that either shell worked. The knowledge base was just a little longer using the back chainer. The decision to use M.1 was primarily based on the differences in rule implementations. Cogito's rules are English-like phrases, whereas, with OPS5, the rules are reminiscent of LISP code, the implementation language. Experts not familiar with LISP have trouble reading and understanding the content of the rules. It is important that the knowledge engineer be able to verify with the expert that the intent of the rule is the same as the coded rule. This is especially crucial if the expert system is giving incorrect advice.

Testing

Testing this expert system was difficult. It was impossible to present it with every possible combination of inputs and evaluate its outputs. The best test we devised was to let the intended users use it in many hypothetical circumstances. If the knowledge base was incomplete, then, in some situations the advice given to the users should be incorrect. Cogito was tested by the Systems Administrator of the Department of Systems and Industrial Engineering, a Professor of Systems and Industrial Engineering, and the Systems Administrator for the Department of Computer Science. They all found Cogito's advice to be complete and correct.

One of our colleagues suggested that the system be tested by 12 random graduate students and that the results be subjected to statistical analysis. We felt this would be unfair, because the system was designed to be used by people with a good knowledge of computer hardware and UNIX software and we only knew of four such people on the University of Arizona campus (the builder of the system and the three testers).

In an effort to bolster our testing we asked several non-expert computer users to try the system. Their evaluations tended to emphasize the difficulties they had using it or making sense of its queries or output, and the extent to which they could "fool" the system with plausible (but nonsensical) inputs. They did not fool this system. So, we did our best to test Cogito, but we were not able to prove that it would always give the correct advice. Testing seems to be a problem with most expert systems.

The best way to test this system would have been to use it to bring up a brand new system. Unfortunately no such system was available. The next best test would have been to shut down an existing system, and rebuild it using Cogito. Unfortunately no one wants to let you shut down their operational system. Recently, however, due to inadequate glue on the heads of our RA81 disk, we had to rebuild our system from the distribution tapes. Cogito helped us. We found a few omissions in Cogito's advice, but no mistakes. It was a big help. We completed the task in about 12 hours.

This experience has reinforced our belief that all expert systems are inadequately tested. There are no quantitative procedures for testing expert systems. Most tests merely involve running a few case studies; they do not exhaust all possibilities. For example, we are confident that Cogito works well for a small VAX 750 system but we cannot be sure that it will work as well for a 730 or a 780.

Appropriateness of Expert System Technology

How can one identify a task that is appropriate for an expert system? First, there must be a human expert who performs that task better than most other people. Second, the task must be one to which the human expert can explain the solution in words, not one that requires the expert to draw a picture to explain what to do. Third, can the problem be solved routinely in a 20 minute, or even a one hour, telephone conversation with the expert? If so, the problem is a good candidate for a personal computer based expert system. If a human would take two days to solve the problem, it is far too complicated for an expert system; if the human gives the answer in two seconds, it is too simple.

Given these criteria, giving advice for bringing up UNIX on a VAX computer was inappropriate for a personal computer based expert system. Bringing up UNIX cannot be done in a one hour conversation with an expert. We think it would take an expert one to two days to do the task. (It took us three months to do it the first time!) The 700 rules of this expert system filled up two floppy disks. We succeeded in making an expert system that worked, but it was hard work. We think a more powerful tool (such as KEE, S.I, ART, or Knowledge Craft) would have been more appropriate.

For further information about Cogito, phone Professor Bahill at (602) 621-6561.

Acknowledgement

We thank Phil Kaslo and Bill Ganoe for testing our expert system.

References

- [1] P. N. Harris, *COGITO: An expert system that gives advice for making and installing UNIX 4.2BSD on VAX-11 series computers*. University of Arizona: master's thesis, 1986.
- [2] *UNIX Systems Manager's Manual*. El Cerrito: USENIX Association, 1985.
- [3] *UNIX Programmer's Manual Reference Guide*. El Cerrito: USENIX Association, 1985.

Access to UNIX Standards

John S. Quarterman

USENIX Representative to the IEEE P1003.1 Committee
usenix!jsq

The IEEE P1003.1 Portable Operating System for Computer Environments Committee is sometimes known colloquially as the UNIX Standards Committee. They have recently published the 1003.1 "POSIX" Trial Use Standard. According to its Foreword:

"The purpose of this document is to define a standard operating system interface and environment based on the UNIX Operating System documentation to support application portability at the source level. This is intended for systems implementors and applications software developers."

Copies are available at \$19.95, with bulk discounts available. To order, call

IEEE Computer Society
(714) 821-8380

Request *IEEE 1003.1 Trial Use Standard* - Book #967.

The Trial Use Standard will be available for comments for a period of about a year. The current target for a Full Use Standard is Fall 1987. IEEE has initiated the process to have the 1003.1 effort brought into the International Organization for Standardization (ISO) arena.

There is a paper mailing list through which interested parties may get copies of drafts of the standard. To get on it, or to submit comments directly to the committee, mail to:

James Isaak
Chairperson, IEEE/CS P1003
Charles River Data Systems
983 Concord St.
Framingham, MA 01701
decvax!frog!jim

Sufficiently interested parties may join the working group. The next scheduled meetings of the working group of the committee are

September 17-19, 1986 Palo Alto, CA
hosts: Amdahl, HP and Sun
December 9-11, 1986 Atlantic City, NJ
same time as X3J11
March 2-6, 1987 Toronto, Ont.

June 8-12, 1987	Phoenix, AZ
	(the week of the USENIX Conference)
September 1987	New Orleans, LA
There is also a balloting group (which intersects with the working group). Contact the committee chair for details.	

Related working groups are:

group	subject	co-chairs
1003.2	shell and tools	Hal Jespersen (Amdahl) Don Cragun (Sun)
1003.3	verification	Roger Martin (NBS) Carol Raye (AT&T)

1003.1 and 1003.2 will meet concurrently in Palo Alto on September 17.

There is frequent discussion of issues related to the various P1003 committees in the Usenet newsgroup *mod.std.unix* (soon to be known as *comp.std.unix*).

The Abstract of the 1003.1 Trial Use Standard adds:

"This interface is a complement to the C Programming Language in the C Information Bulletin prepared by Technical Committee X3J11 of the Accredited Standards Committee X3, Information Processing Systems, further specifying an environment for portable application software."

The liaison from X3J11 (sometimes known as the C Standards Committee) to P1003 is

Don Kretsch
AT&T
190 River Road
Summit, NJ 07901

A contact for information regarding publications and working groups is

Thomas Plum
Vice Chair, X3J11 Committee
Plum Hall Inc.
1 Spruce Avenue
Cardiff, NJ 08232

There is frequent discussion of X3J11 in the Usenet newsgroup *mod.std.c* (that newsgroup will eventually be known as *comp.std.c*).

* POSIX is a trademark of the IEEE.

The /usr/group Standard is the principle ancestor of P1003.1:

/usr/group Standards Committee
4655 Old Ironsides Drive, Suite 200
Santa Clara, CA 95054

The price is still \$15.00.

Heinz Lycklama of Interactive Systems Corp. is the /usr/group institutional representative to P1003.1.

The System V Interface Definition (The Purple Book): this is the AT&T standard and is one of the most frequently-used references of the IEEE 1003 committee.

System V Interface Definition, Issue 2

Select Codes 320-011 (Volume 1) and 320-012 (Volume 2) or Select Code 307-127 (both volumes).

AT&T Customer Information Center
2833 North Franklin Road
Indianapolis, IN 46219
1-800-432-6600, operator 77.

The price is \$37 for each volume or \$57 for the pair. Major credit cards are accepted for telephone orders: mail orders should include a check or money order. Previous SVID owners should have received a discount coupon to upgrade to Release 2 for only \$37.

Volume 1 is essentially equivalent to the whole previous SVID; Volume 2 is mostly commands and a few add-ons (e.g. *curses*). A third volume is expected

in the last quarter of 1986 to cover new items in System V Release 3, such as streams and networking. There may be an upgrade discount similar to the previous one. A draft copy is reputed to be available now to source licensees.

The *X/OPEN Portability Guide* (The Green Book) is another reference frequently used by IEEE 1003. X/OPEN is "A Group of European Computer Manufacturers" who have produced a document intended to promote the writing of portable facilities. (They now have member computer manufacturers from outside Europe.) Their flyer remarks (in five languages), "Now we all speak the same language in Europe."

The book is published by

Elsevier Science Publishers
Book Order Department
P.O. Box 211
1000 AE Amsterdam
The Netherlands

or, for those in the U.S.A. or Canada:

Elsevier Science Publishers Co Inc.
P.O. Box 1663
Grand Central Station
New York, NY 10163

The price is Dfl 275,00 or US \$75.00. According to the order form, "This price includes the cost of one update which will be mailed automatically upon publication."

Book Review

The UNIX C Shell Field Guide by Gail Anderson and Paul Anderson
(Englewood Cliffs, NJ: Prentice-Hall, Inc., 1986) \$23.95

Reviewed by Marc D. Donner

ucbvax!ibm.com!donner

Back when I was first learning UNIX the conventional wisdom held that you used the C shell as your login shell because it had aliases and history but that you wrote all of your shell scripts in the old Bourne shell because it had a simpler interface. More conventional wisdom held that you only wrote shell scripts if they were short because the performance was so terrible, the only exceptions being things that were executed rarely or that had to be portable as files without recompiling, for example the *rc* script.

The UNIX C Shell Field Guide is a lengthy tutorial introduction to the C Shell. The recent fame of our favorite operating system has resulted in an explosion of publications offering to teach us things about UNIX. Despite the fact that I rarely learn anything from any of them, I regularly purchase all the latest offerings in the hope that I will find some great insight offered up. Let's look at this new book and see what it has to offer.

This is a tutorial, so experienced hackers can go back to sleep. The English is reasonably good, a definite plus in a tutorial. The prose is warm and friendly, something that will make many a novice user feel good, but it is a bit too cutesy for my taste. I found my hackles rising several times in response to phrasing that I found patronizing. This is a clear sign that I am not part of the intended market for this book.

Now that we have identified the intended audience, novices, let's take a look at it from their perspective. As a novice's tool it is fairly daunting. The book is almost an inch thick, holding 374 pages. The table of contents is ten pages long, making it difficult to grasp the overall structure and content of the book looking at it. The index is quite good, and it is the last section of the book, to the authors' credit. All too often the publisher appends many pages of trash at the end of a trade book like this, making the job of finding the index so difficult as to destroy the utility of the book. I won't dignify their pun of replacing the heading for the X section of the index with IX with any comment.

The authors took the three Jesuit principles of teaching to heart: repetition, repetition, repetition. Each important topic is covered several times, though their presentation would be much better if the cross referencing were better. The overview section on aliases should refer the user to the chapter and page of the in-depth section on aliases, for example.

A tutorial should provide the reader with some simple examples to try, and this one does that quite well. Some of the examples are a bit contrived, detracting from their instructional and mnemonic value. The explanation associated with the examples is fairly spotty. Sometimes it is clear and informative, and sometimes it contains flat statements of fact unilluminated by any attempt at motivation. For example, when exhibiting a call to *find* with the argument '*()*' to indicate where to substitute the path name, the book simply states the meaning of the argument, with no motivation or apology. Perhaps it isn't fair to pick on *find*, widely known to be the most obnoxious command in UNIX, but maybe the authors shouldn't have used it in an example.

There is one fairly serious technical flaw in this book. The most important thing they could teach about the C Shell or any shell is the environment state problem. It is important to emphasize that a shell script, executing in a child process, cannot modify the state of the parent shell. The only way that a shell script can modify the parent environment is to be invoked by the source command. This restricts the script to the shell language interpreted by the parent or login shell. This problem is so important and so confusing to novices that it almost merits a chapter of its own.

The most important thing that I look for in a tutorial is some help in organizing the material so that I can structure it and remember it. This is best done by identifying and highlighting important general principles and returning to them from time to time in the exposition and in the examples. This book's greatest weakness is its shortage of general organizing principles, though this drawback might better be ascribed to the subject matter. The book is full of interesting details, but short on compelling generalizations or mnemonics.

;login:

Most chapters have a summary of key points and a collection of hints and cautions, a wonderful idea. Unfortunately, the things that are warned about in the hints and cautions sections are not always the important things. The problem with environment modification mentioned above doesn't make it to the hints and cautions section, for example. Overall, this book doesn't do a very good job of warning the user about the dangers of using the C Shell. The C Shell is very powerful, but also very dangerous. When I taught a bunch of novices how to use it a while ago most of my presentation was concerned with hazardous things to watch out for ... the power and function are sufficiently exhibited in the man pages.

Does the world need another book about the C Shell? Does the world need this one? That's difficult to answer. This book has its charm, but I wonder about its utility. If I were asked to design a book about the C Shell, it would be quite a bit different from this one. It would be no more than 100 pages long and it would be primarily reference material, organized for easy reference. The recent AT&T C handbook is an outstanding example of how to do this well. If it had a tutorial section, it would be much shorter than this one and would emphasize general principles through examples, rather than details.

:login:

The USENIX Association Newsletter

Volume 11, Number 6

November/December 1986

CONTENTS

The Future of ;login:	3
Software Distribution Tape 86.2	4
4.3BSD Manual Shipping Delay	4
Personalizing the Impersonal	5
<i>Dave Taylor</i>	
Future Meetings	12
Hindsight is 20/20	13
Washington D.C. Meeting Program	14
WeirdNIX . . or Destructive QA of a Standard	27
SALE! SALE!! SALE!!!	28
Why Source Licenses?	29
Ten Years Ago in ;login: (a.k.a. UNIX News)	29
4.3BSD UNIX Manuals	30
4.3BSD Manual Reproduction Authorization and Order Form	31
Summary of Board of Directors' Meeting in Atlanta, June 9-10	32
The 'Stargate' Project	33
Atlanta Videotapes Available	33
Future Workshops	33
Local User Groups	34

The closing date for submissions for the next issue of ;login: is January 5, 1987



THE PROFESSIONAL AND TECHNICAL UNIX® ASSOCIATION

Personalizing the Impersonal

*and Other Tales of Communication in the
Computer Age*

Dave Taylor

Hewlett-Packard Laboratories
hplabs!taylor

Introduction

One of the greatest drawbacks of computer-based communication is the lack of presence of the medium. When we talk to someone in person, we can see them, receiving cues from their clothing, posture, facial expressions, and their very physical presence. On the telephone we also receive considerably more information than just the words, including volume, pitch and intonation.

On computers, however, there is nothing but the words themselves. This leads to a surprising number of problems in communicating, mostly due to our language having evolved to be spoken, rather than written.

To combat this, a number of interesting techniques are becoming commonplace, most of which are attempts to personalize the medium. Herein we'll present a brief overview of the most common attempts, including overt hostility, flaming, and various stylistic topics, and discuss motivational factors.

Simulating Visual Cues

When talking to someone in person, one of the most useful sources of information is body language. Many books have been written on interpretation of visual cues. Such cues are not available with computer-based communications, however, and a number of alternative ways of simulating them have arisen. Perhaps the most noticeable is the ubiquitous smiley-face ":-)" which, when viewed by tilting your head to the left, actually looks like a face. This is used to represent a specific cue, as are a number of variations.

- :-) This is used at the end of a passage of text to indicate that the previous was either sarcastic or should be taken as a joke. For example, it's common to have phrases like "If this was a QUALITY system... :-)" where the understood meaning is that the phrase is said in jest, not seriously.
- ;-) (A "winking" smile) This is similar in meaning to the previous, but is used as a stronger expression of levity. It's also quite commonly used to indicate lewd, but lighthearted, comments. For example, "So, want to sleep at my place when you're in town? ;-)" Again, the understood meaning is that it's all in fun.
- :-(> (Unhappy face) This symbol represents unhappiness, and usually is used to emphasize a bad event occurring, as in: "and our SYSOP told me we don't DO backups anymore!! Two months worth of work LOST! :-("

That these symbols have "evolved" purely because of the need to transmit such cues is an interesting example of the communicative value of non-verbal aspects of language.

Simulating Verbal Cues

Some members of the electronic community feel that the caricatures are "juvenile," however, and they add cues in a manner more akin to a playwright (the previous can be typified as a cartoonist approach). Their communications are punctuated by phrases expressing the appropriate verbal and visual cues:

"Trust me. This'll work! *evil laugh*"

Obviously, this approach tends to convey more information than the icons, with the penalty of increased verbosity.

The stressing of individual words or phrases (verbal cues) also shows the varied approaches taken; by far the most common cue being all uppercase letters. There are a large number of different techniques currently being used:

"...and then SUE walked in to the room. I just *couldn't* believe it.
Of all the people to show up, >Sue< was the last one I was expecting!
It was \fIreally\fP mind-boggling!"

The last example, the word being surrounded by "\fI" and "\fP" is worth special note – it is the sequence in a popular text formatting system by which one generates italics. It's as if the author assumes that the reader can emulate the formatting system. (Another example of this is more common – underlined phrases are delimited by leading and trailing underscores, as in _this phrase_. A different text formatting system uses this notation to delimit a passage to be underlined. As with the other form, it's assumed that the person can interpret the text accordingly).

Presentation Styles

As anyone who took a high school typing class will know, oftentimes the form is as important as the content of a message. Computer-based communications systems are no different and a number of different presentation styles have evolved to allow the authors to express their individuality.

The most common is to have nonindented paragraphs and ragged right margins. Other popular formats are to add indentations (and they, of course, vary from two characters up to ten), to indent the entire text, to have "inverse paragraphs" (that is, where the first line is flush left and the rest of the lines of a paragraph are indented, traditionally used for expository lists) and to have both left and right margins aligned.

The last, left- and right-justified text, can only be accomplished through the use of intermediate programs. This is intriguing because these authors are willing to place an artificial barrier between themselves and the medium in which they are writing, in the interest of a unique presentation style. It is akin to writing a letter to someone in pencil and then overwriting each character in ink.

Another common presentation style, due to the high rate of quoting present, is exemplified by:

Message from Person B:

> Person A writes:
> So I told my manager that their expectations were artificially high
Really? You actually had the courage to tell him that?
> and that if they really want to have the project
> complete on time that they should stop giving me new people to break in
snicker
> and just let me GET ON WITH THE JOB!
Right on!

What's interesting about this is that Person B has decomposed the original message into "conceptual units" and commented on them individually. Here's a further example:

Message from Person C:

> Person A writes:
> So I told my manager that their expectations were artificially high
I've found it common to have managers that don't
understand the reality of us folk on the front lines.
Good stuff. C.

Notice that the original meaning has been lost. In fact, readers who haven't seen the original might easily misconstrue the quote to be a general criticism of management by A, when it was not meant to be at all.

This is, paradoxically, due to the lack of a proper context for the quote. It seems to be a general problem with the sequential soliloquy dialogues of computer conferencing systems – authors are expected to minimize the quoting of others (to avoid "information overload") but at the same time must ensure that they don't quote others out of context.

Writing Styles

Along with presentation styles, and methods of quoting, different authors are finding that specific recognizable styles are also aids towards personalizing their communications.

A number of examples exist, from the lack of capitalization of 'i' when used as a pronoun to the complete lack of capitalization of names, to the use of ellipses to end sentences.

There are many interesting examples of these, one of which exhibits a number of these styles:

Message from Person X:

Nice idea, but it should be more than just a *mail* transport system -- it should include other forms of (only electronic?) communications as well.

Provide one [powerful, consistent] interface to the world, and it will be SIMPLE. AND that means it'll be useful.

That's how i'd do it, at least.

There are a number of interesting occurrences here, including the use of parenthetical remarks to suggest alternative interpretations of phrases, and the very unusual (in English prose) use of square brackets ("..one [powerful, consistent] interface..") to delimit further meaning to the phrase.

Also notice that other rules of English prose are ignored with regularity, such as beginning a sentence with a conjunctive ("AND that means...") or, as already noted, not capitalizing proper names.

It is this deliberate alteration of the language normally used for communication that is the tie-in with the personalization of the medium.

Signatures

Perhaps one of the most interesting areas of personalization of communication is the message closings, the individual signatures. Not surprisingly, it's common to see messages where the closing section is actually considerably *longer* than the body.

In this inherently impersonal medium, the signature is the one place, more than any other, where authors freely exhibit their individualism.

Electronic signatures may be broken into the following sections: closing salutations, the "signature" name, and disclaimers.

Closing salutations in more traditional mail are typically "Sincerely," or "Affectionately yours," etc. In electronic communications, by contrast, closing salutations are much more varied:

Take care, all.

Share and enjoy,

May the light of love,
fill your life

cheers!

From the Crow's Nest,

Note that some are indicative of the author personally caring about the unknown reader (presumably with the assumption that if the author exhibits concern about the reader, then the reader will, in return, care about the

:login:

author), another is a "generic" salutation (with overtones of a particular lifestyle, however, e.g. the social drinking of alcohol) and the next is a self-label based on their place of employment (identifying with a group).

It's also interesting to note that certain conferences tend to have groups of mutually supportive members. The examples above are all from such a tight knit group.

If we examine a group with a more technical orientation, therefore, the closing salutations are, as expected, more technically oriented:

From the tty of,	Guru-in-Waiting,
Thanks in Advance,	In Real Life,
Any Ideas?	

These are requests for information and less personal altogether. There is still the tendency to personalization, though, but it is much less apparent than in the more social groups.

Interestingly, technical groups have a fair number of unsigned articles, where there is a message body only. This almost never occurs in the more social groups.

Lacking the flair and individuality of handwriting, the closing name tends to be "permuted" to personalize them instead:

_J	-- John Doe
-- jd --	John "Killer" Doe
John "I don't use it" Doe	

As a lot of messages originate while the author is at work, disclaimers are also common as postscripts:

"Disclaimer: The above ideas are mine--I don't yet know who agrees with me."

"(The opinions above are my own, colored by my experiences, education and experimentation, and should not be confused with anyone else, unless they wish to share them, and in so, enrich us both)"

Notice that even in a standard section like a disclaimer, there is a large degree of personalization, to the point that the original intent of the disclaimer (to ensure that readers don't misconstrue the message as representative of the organization the author is a member of) has been lost entirely.

(I expect to have people contact me saying "That's *my* signature!" or something equivalent. This occurrence will show the degree to which the personalization of the medium has succeeded – there are thousands of people who send electronic messages every day, and yet we still recognize our own prose!)

Using Fictitious Names

I will only touch on this topic since it has been covered in detail elsewhere, but it's interesting to note that there is a fair amount of self-enforced anonymity on conferencing systems – people will use names like "pooh" or "phoenix" (often to avoid name recognition when met in person, but that's another topic entirely). In more socially-oriented discussion forums, there is more likelihood that someone will be "masquerading" as another, perhaps fictional person.

Also common is for people to "represent" fictitious organizations on the system, occasionally so as not to discredit where they work. Examples are a person from NASA having "The Home for Retired Hackers" as their organization, or someone from Sun Microsystems having their organization listed as "Fictional Reality, Ltd."

Hostility

A number of interesting articles have been written on the high level of emotion on conferencing and electronic communications systems, but it's worth another look from the perspective of personalizing an impersonal medium.

A typical electronic message is considerably more flamboyant and "charged with emotionalism" than the equivalent message in a more traditional medium (e.g. in person). The main reason for this is the desire to personalize the medium. Authors of messages are often judged purely on the content of the message rather than on the numerous other possible information cues, so the message has to be significant.

Consider a typical sequence of electronic mail messages (we're joining the conversation in mid-stream. Ignore the actual details of what is being discussed and read the tone and extreme emotion of the messages):

Message From Person 1

Subject: Re: termdb and so on...

>-- PARTICULARLY when Company X then doesn't include the sources to the
>termdb descriptions...

Give me a break...What the hell more do you want?

...
Real difficult. Termdb just beats the pants off the old system;
the sooner it disappears the better.

Notice the occurrences of inflammatory phrasing when a less dramatic tone would have sufficed. Particularly notice the overt hostility of the phrase "What the hell more do you want?" The number of spelling/grammatical mistakes is also surprisingly high – including the misspelling of *particularly*. Finally, notice the number of occurrences of colloquialisms, including "step forward," "giving a break," "beats the pants off," and so on.

Reply from Person 2:

>Give me a break...What the hell more do you want?

"what the hell more do you want?" Sounds like a nasty attitude
<person 1>...So what's your problem <person 1>? Why are you jumping
down the throat of some frustrated user? This is not a good sign of
your understanding of the Customers' needs. You do understand that
(no customers = no jobs) don't you? You know, we really can walk across
the street now, and we just may if attitudes like yours don't improve real quick.

Again, notice the hostility, and especially the aggressive physical threat against the previous author "...we really can walk across the street now, and we just may..."

Compare the last to another person's response to the same original message:

Reply from person 3:

> Give me a break...What the hell more do you want?

...
> Real difficult. Termdb just beats the pants off the old system;
> the sooner it disappears the better.

FLAME ON[†]:

[†] Flames are particularly hostile remarks. See the section entitled 'Flaming' for a further discussion of this phenomenon.

:login:

Unfortunately, not all of us happen to have SVR3, nor do we have dbcmp(1) or anything similar. But that's not my biggest beef...

...I do *not* like having to clean up after it throws up on my filesystem, and I definitely do not consider that a "huge step forward in design".

Flame off.

By the way, if anyone can send me an untic program, I would be much obliged...

This time notice the fascinating transition from overt hostility to a request for helpful information ("By the way, ..."). It's almost as if the phrases "Flame On" and "Flame Off" completely release the author from having to be civil, since the reader "understands" that the section is an aberration.

Finally, we'll end this listing of messages with another from the author of the original:

A reply from Person 1:

>"what the hell more do you want?" Sounds like a nasty attitude <person 1>

I apologise for the snipey remarks.

>So what's your problem <person 1>? Why are you jumping down
>the throat of some frustrated user?

Because the frustrated user was ranting and raving. If I flame
about you, you have every right not to be civil to me.

>This is not a good sign of your understanding of the Customers' needs.

I understood exactly what the needs were. I just didn't like
how they were conveyed, and the misinformation along with it.

The user gets to yell and scream, but I don't? Feh.

If you talk reasonable, so will I.

Based on the last message, we could argue that to some extent the conferencing system is a self-moderated group, with people being publicly embarrassed by others and then apologizing to the entire group, but I don't think that this is the case at all. A close reading of the last message seems rather to indicate that the author is saying "I'll pretend to be sorry, but I'll phrase it in such a way as to vindicate myself and shift blame to the original author." Consider the meaning of the very last sentence relative to the rest of the message... "If you talk reasonable, so will I."

This rather long example is a particularly interesting one, as it not only demonstrates a number of the areas I've been discussing, including extensive quoting, adding verbal cues and so on, but also is an excellent sample of the flamboyant emotionalism of the medium.

Hecklers

While the previous problem seems widespread in computer-based communications, another, more glaring, example of this occurs occasionally due to the inability to suppress inflammatory interjections by hecklers. In an interpersonal situation it is easy to have someone forcibly removed if they get overly abusive, but in a distributed environment, there is no way other than peer pressure, which is oftentimes too difficult to coordinate (especially if the consensus is to ignore the offender).

Before we consider why this sort of behavior might tempt someone, let's see an example:

Message from <person X>

Subject: flames were called for

;login:

I have read a few notes on the boards or should I say flames of sorts about my posting about <x> etc. For those of you who didn't read my last note, the one that appears right before this one, then please do, and I welcome any E-mail flames or non-flames you might have. I think in reality its a subject that people are to afraid to talk about openly...

I posted my feelings about the subject can you?

Not only is this a very inflammatory message to include in an international conference, but it's also a fine example of the what causes this type of antisocial behavior. It's the same motivational factor that causes authors to personalize the medium, just exaggerated: the need to "fit in" and be accepted and appreciated by peers.

Specifically, on a system where one receives electronic mail from people one has never met, a simple measure of popularity is the amount of mail one receives and the number of times one is referred to in other messages. With hecklers, it seems that the context in which they are referred to is irrelevant.

To support this, consider the phrases "I welcome any E-mail flames or non-flames that you might have" and "I posted my feelings about the subject can you?" surrounding statements that deliberately provoke the reader (e.g. "I think that ... its [sic] a subject that people are to [sic] afraid to talk about...").

From this perspective it becomes obvious why heckling can be a successful strategy: to receive lots of electronic mail. This is why a number of conferencing systems are plagued with this type of antisocial behavior.

Flaming

Another area worthy of note is the concept of flaming. In the common vernacular it's used as a way of apologizing in advance for could be construed as overly hostile and abusive writing. The message presented previously, with the "Flame On/Flame Off" bracketing is quite typical, and is an attempt to add the ability to get angry without public embarrassment.

In normal life we all have moments of hostility, often towards others. A typical reaction is to go to a third party and complain (either an involved person (e.g. a boss) or an interested party (e.g. a spouse or friend)) or to confront the cause. If the original incident is somehow aired publicly, however, the injured party will feel that the response should be broadcast in a similar fashion. This is most noticeable in the mud-slinging of politicians during elections.

On the other hand, if the medium is a persistent one that others could possibly turn against the injured party, an attempt is made to somehow excuse the reply, to somehow let it be known that the following reaction is temporary and should not be construed as indicative of typical behavior of the author.

This is the role that the flaming appears to have taken. It is a method by which authors can write extremely hostile and inflammatory remarks, oftentimes littered with obscenities, without having their names "besmirched."

Positive Aspects

While these attempts to personalize the medium might seem to make it harder to read and understand the information being transmitted, there is an important benefit that hasn't been mentioned.

Electronic communications systems are the first form of communication where people are more akin to "expressers of ideas" rather than individuals with the idiosyncratic quirks we exhibit in person. That isn't to say that the medium is totally impersonal – we've already seen this isn't the case – but when one reads a message from another, the only information available is the message itself. There are no external cues to pick up on, such as age, race, sex, appearance, and such.

This has some significant ramifications in the treatment of the handicapped in our society. Currently, most handicapped people, either "visibly" handicapped or otherwise, are treated differently from those that don't exhibit any disabilities. Subsequently, when handicapped persons desire to communicate their ideas to the rest

of a group, their ideas and thoughts are treated differently too (and too often discounted, as if a physical disability automatically means the person is mentally impaired as well).

On the computer-based communications systems, though, we've seen that all the common cues are simulated through various techniques. This directly translates to people being equal regardless of their physical or mental state until the message is read. The act of communication is completely on the message itself, and nothing else.

As a direct consequence of this, some members of conferencing systems have what would ordinarily be considered serious impairments yet they can participate without anyone else knowing they are handicapped. (This is not to say that all handicapped people wish they weren't, but certainly they desire to be treated as equals, as do all minorities.)

Finally

This is only a brief overview of the different ways by which people have begun to personalize this medium, and as the computer systems become more powerful (being able to transmit graphics, and font information, like *italicize* or **bolden**) we are sure to see yet more interesting and innovative ways to personalize electronic communications.

Note

The predominant source for the information presented in this article has been the Usenet, an international distributed conferencing system running on the UNIX® operating system, the conferencing system on the "Well" (The Whole Earth 'Lectronic Link) and private electronic mail.

Specific conferences that I'd like to acknowledge as being used for the examples are *net.singles*, a social group that discusses topics relevant to being single in our society, *net.unix*, a highly technical conference oriented towards questions and discussions about the UNIX operating system and related software, and *sexuality*, a forum discussing sexuality in our society on the Well.

The book *The Network Nation*, by Starr Roxanne Hiltz and Murray Turoff (Addison Wesley, 1978) explores a number of these topics in greater detail, and was helpful in confirming a number of observed behaviors and the probable motivations causing them.

Hindsight is 20/20

absolutely anonymous[†]

The sad plight of computer professionals rears its ugly head each time a new salary survey emerges. More and more nowadays we find that lawyers, doctors, and even accountants are heading up the salary scale. Computer people? "Well," many say, "My brother works at a bank. He makes pretty good money."

How did the world's most honored and technically challenging profession end up out of first place? We didn't use our heads in 1957; that's how. Let's do a comparison with the medical field.

The medical field deals with a rather complex machine, the human body. Doctors make life and death decisions. They administer drugs, perform surgery, diagnose diseases, and in general try to maintain the machine in good working order.

Oh, doctors have a few good perks: good salary, prestige, a place in the community. Doctors make **BIG MONEY**. They work hard, though, they tell us. They have to get up early in order to perform surgery (or to deliver babies!). They have to spend many years in school. They have to put up with a general public which is often abusive and unappreciative.

How different is our field? We deal with a complex machine. If we accidentally kill it, it's our job to fix it. If we can't fix it, we don't get paid! Doctors get paid no matter what. We don't administer drugs but we perform surgery, diagnose problems, and in general maintain machines in 100% working order; 99% is not enough. Have you ever heard a field engineer say, "George, I'm afraid we're going to have to amputate your floating point accelerator; it has cancer." Of course not. How about, "Harry, I'm afraid your payroll program has but six months to live. You better get its affairs in order."

Do we make **BIG MONEY**? I don't. I don't know many who do. Do we work hard? Of course we do. Do we get up early (or stay up late)? You bet. Years of school? Lots of computer jockeys with Ph.D.'s spent more than a decade in school. Put up with an abusive public? Ha!

WHERE DID WE GO WRONG? I'm glad you asked that. Firstly, we did not point out how complex and extraordinary our machines are. You know, not just anyone can plug in a new Multibus disk controller. Chips? We should have guarded them like the King's Orb. We actually ENCOURAGED people to learn how to design, program, and repair our lovely machines. What a screw-up. We should have raised machines at least to the level of sanctity of the human body, maybe as high as the gods themselves.

Doctor schools have controlled admissions. There's never enough doctors. In a supply/demand economy, we flooded the world with supply. Dumb, dumb, DUMB. Did we license programmers? Oh no. Anyone can program; just read any LOGO book. Go to your local high school; are the kids practicing with new drugs (therapeutic drugs, you knew what I meant) or practicing performing surgery on their siblings? Of course not. They're too busy shooting aliens on their Apple IIe.

So, like it or not, we're doomed to be also-rans in the world of professionals. There's nothing we can do now short of hoping for a tremendous EMI pulse from some solar flare that will knock out every computer from here to the mountain in Boulder. We can't just shoot every BASIC or COBOL programmer on the earth. We've lost it.

Maybe we should try again in Gallium-Arsenide. Think anyone would believe that it's actually magic and requires an applied wizardry degree?

[†]The following anonymous piece of humor arrived in my e-mail. -PHS

;login:

The USENIX Association Newsletter

Volume 12, Number 1

January/February 1987

CONTENTS

Call for Papers – Summer 1987 USENIX Conference	3
;login: has New Technical Editor	4
Ten Years Ago in ;login: (a.k.a. UNIX News)	4
How To Write a Setuid Program	5
<i>Matt Bishop</i>	
Call for Papers – 4th USENIX Computer Graphics Workshop	12
An Overview of the Sprite Project	13
<i>J. Ousterhout, A. Cherenson, F. Douglis, M. Nelson, and B. Welch</i>	
Book Review: The C Programmer's Handbook	18
<i>Marc D. Donner</i>	
Standards	19
Call for Papers – System Administration Workshop	20
Future Meetings	21
Atlanta Videotapes Available	21
4.3BSD UNIX Manuals	22
4.3BSD Manual Reproduction Authorization and Order Form	23
Proceedings of 3rd Graphics Workshop Available	24
Publications Available	25
Human – Computer Interaction Conference	25
Local User Groups	26

The closing date for submissions for the next issue of ;login: is February 27, 1987



:login:

Call For Papers

Summer 1987 USENIX Conference

June 8-12, Phoenix, Arizona
Civic Plaza – Convention Center

Abstracts are being accepted from individuals wishing to present papers at the 1987 Summer USENIX Conference. Abstracts should be 250-750 words long, emphasizing what is new and interesting about the work. The final paper should be 8-12 pages when typeset.

Suggested topic areas include, but are not limited to:

- Kernel enhancements, measurements, etc.
- Programming languages and environments.
- UNIX in the office environment.
- Standards and portability.
- New mail systems (e.g., X.400-based systems or user interfaces incorporating new interface paradigms).
- Applications, especially unusual ones such as computer aided music, factory automation, etc.
- Security.
- UNIX vs. the naive user.
- Workstations: comparisons, experiences, trends.
- Beyond UNIX – what next?

Vendor presentations should contain substantial technical information and be of interest to the general community.

All abstracts will be due by February 20, 1987. Electronic submissions to *ucbvax!phoenix* or *phoenix@Berkeley.EDU* are preferred. All authors whose abstracts are accepted must submit a paper by the publication deadline or they will forfeit their talk. We currently plan to encourage electronic submissions of final papers in *troff* format using the *-ms* or *-me* macros and the *tbl*, *eqn*, and *pic* preprocessors.

Relevant dates:

20 February	Abstracts due
16 March	Notifications to authors
13 April	Final papers due
10-12 June	Conference program

The program committee consists of:

Eric Allman, Britton Lee (chair)
Greg Chesson, Silicon Graphics
Sam Leffler, Pixar
Jay Lepreau, University of Utah
John Mashey, MIPS

John Quarterman
Dennis Ritchie, AT&T Bell Laboratories
David Taylor, Hewlett-Packard
Chris Torek, University of Maryland

For additional information regarding the program, contact:

Eric Allman
Britton Lee
1919 Addison Suite 105
Berkeley, CA 94704
(415) 548-3211 (work) (415) 843-9535 (home)
ucbvax!eric eric@Berkeley.EDU

Please include your network address, if available, with all correspondence. This should be an Arpanet address or a UUCP address relative to a well known host; if in doubt, start from *mrvax*, *ucbvax*, *decvax*, or *seismo*.

How To Write a Setuid Program

Matt Bishop

Research Institute for Advanced Computer Science
NASA Ames Research Center
Moffett Field, CA 94035

ABSTRACT

UNIX systems allow certain programs to grant privileges to users temporarily; these are called setuid programs. Because they explicitly violate the protection scheme designed into UNIX, they are among the most difficult programs to write. This paper discusses how to write these programs to make using them to compromise a UNIX system as difficult as possible.

Introduction

A typical problem in systems programming is often posed as a problem of keeping records [1]. Suppose someone has written a program and wishes to keep a record of its use. This file, which we shall call the *history file*, must be writable by the program (so it can be kept up to date), but not by anyone else (so that the entries in it are accurate). UNIX solves this problem by providing two sets of identifications for processes. The first set, called the *real* user identification and group identification (or UID and GID, respectively), indicate the real user of the process. The second set, called the *effective* UID and GID, indicate what rights the process has, which may be, and often are, different from the real UID and GID. The protection mask of the file which, when executed, produces the process, contains a bit which is called the *setuid* bit. (There is another such bit for the effective GID.) If that bit is not set, the effective UID of the process will be that of the person executing the file; but if the setuid bit is set (so the program runs in *setuid mode*), the effective UID will be that of the owner of the file, not that of the person executing the file. In either case, the real UID and GID are those of the person executing the file. So if only the owner of the history file (who is the user with the same UID as the file) can write on it, the setuid bit of the file containing the program is turned on, and the UIDs of this file and the history file are the same, then when someone runs the program, that process can write into the history file.

These programs are called *setuid programs*, and exist to allow ordinary users to perform functions which they could not perform otherwise. Without them, many UNIX systems would be quite unusable. An example of a setuid program performing an essential function is a program which lists the active processes on a system with protected memory. Since memory is protected, normally only the privileged user *root* could scan memory to list these processes. However, this would prevent other users from keeping track of their jobs. As with the history file, the solution is to use a setuid program, with *root* privileges, to read memory and list the active processes.

This paper discusses the security problems introduced by setuid programs, and offers suggestions on methods of programming to reduce, or eliminate, these threats. The reader should bear in mind that on some systems, the mere existence of a setuid program introduces security holes; however, it is possible to eliminate the obvious ones.

Attacks

Before we discuss the ways to deal with the security problems, let us look at the two main types of attacks setuid programs can cause. The first involves executing a sequence of commands defined by the attacker (either interactively or via a script), and the second, substituting data of the attacker's choosing for data created by a program.

In the first, an attacker takes advantage of the setuid program's running with special privileges to force it to execute whatever commands he wants. As an example, suppose

an attacker found a copy of the Bourne shell *sh(1)*[†] that was setuid to *root*. The attacker could then execute the shell, and – since the shell would be interactive – type whatever commands he desired. As the shell is setuid to *root*, these commands would be executed as though *root* had typed them. Thus, the attacker could do anything he wanted, since *root* is the most highly privileged user on the system. Even if the shell were changed to read from a command file (called a *script*) rather than accept commands interactively, the attacker could simply create his own script and run the shell using it. This is an example of something that should be avoided, and sounds like it is easy to avoid – but it occurs surprisingly often.

One way such an attack was performed provides a classic example of why one needs to be careful when designing system programs. A UNIX utility called *at(1)* gives one the capability to have a command file executed at a specified time; the *at* program spools the command file and a daemon executes it at the appropriate time. The daemon determined when to execute the command file by the name under which it was spooled. However, the daemon assumed the owner of the command file was the person who requested that script to be executed; hence, if one could find a world-writable file owned by another in the appropriate directory, one could run many commands with the other's privileges. Cases like this are the reason much of the emphasis on writing good setuid programs involves being very sure those programs do not create world-writable files by accident.

There are other, more subtle, problems with world-writable files. Occasionally programs will use temporary files for various purposes, the function of the program depending on what is in the file. (These programs need not be setuid to anyone.) If the program closes the temporary file at any point and then reopens it later, an attacker can replace the

[†] The usual notation for referencing UNIX commands is to put the name of the command in italics, and the first time the name appears in a document, to follow it by the section number of the *UNIX Programmers' Manual* in which it appears; this number is enclosed in parentheses. There are two versions of the manual referred to in this paper, one for 4.2BSD UNIX [2], and one for System V UNIX [3]. Most commands are in the same section in both manuals; when this is not true, the section for each manual will be given.

temporary file with a file with other data that will cause the program to act as the attacker desires. If the replacement file has the same owner and group as the temporary file, it can be very difficult for the program to determine if it is being spoofed.

Setuid programs create the conditions under which the tools needed for these two attacks can be made. That does not mean those tools will be made; with attention to detail, programmers and system administrators can prevent an attacker from using setuid programs to compromise the system in these ways. In order to provide some context for discussion, we should look at the ways in which setuid programs interact with their environment.

Threats from the Environment

The term *environment* refers to the milieu in which a process executes. Attributes of the environment relevant to this discussion are the UID and GID of the process, the files that the process opens, and the list of environment variables provided by the command interpreter under which the process executes. When a process creates a subprocess, all these attributes are inherited unless specifically reset. This can lead to problems.

Be as Restrictive as Possible in Choosing the UID and GID

The basic rule of computer security is to minimize damage resulting from a break-in. For this reason, when creating a setuid program, it should be given the least dangerous UID and GID possible. If, for example, game programs were setuid to *root*, and there were a way to get a shell with *root* privileges from within a game, the game player could compromise the entire computer system. It would be far safer to have a user called *games* and run the game programs setuid to that user. Then, if there were a way to get a shell from within a game, at worst it would be setuid to *games* and only game programs could be compromised.

Related to this is the next rule.

Reset Effective UIDs Before Calling exec[‡]

[‡] Exec is a generic term for a number of system and library calls; these are described by the manual pages *exec(2)* in the System V manual and *execve(2)* and *exec(3)* in the 4.2BSD manual.

:login:

Resetting the effective UID and GID before calling *exec* seems obvious, but it is often overlooked. When it is, the user may find himself running a program with unexpected privileges. This happened once at a site which had its game programs setuid to *root*; unfortunately, some of the games allowed the user to run subshells from within the games. Needless to say, this problem was fixed the day it was discovered!

One difficulty for many programmers is that *exec* is often called within a library subroutine such as *popen(3)* or *system(3)* and that the programmer is either not aware of this, or forgets that these functions do not reset the effective UIDs and GIDs before calling *exec*. Whenever calling a routine that is designed to execute a command as though that command were typed at the keyboard, the effective UID and GID should be reset unless there is a specific reason not to.

Close All But Necessary File Descriptors Before Calling exec

This is another requirement that most setuid programs overlook. The problem of failing to do this becomes especially acute when the program being *exec*'ed may be a user program rather than a system one. If, for example, the setuid program were reading a sensitive file, and that file had descriptor number 9, then any *exec*'ed program could also read the sensitive file (because, as the manual page warns, “[d]escriptors open in the calling process remain open in the new process ...”).

The easiest way to prevent this is to set a flag indicating that a sensitive file is to be closed whenever an *exec* occurs. The flag should be set immediately after opening the file. Let the sensitive file's descriptor be *sfd*. In both System V and 4.2BSD, the system call

```
fcntl(sfd, F_SETFD, 1)
```

will cause the file to close across *exec*'s; in both Version 7 and 4.2BSD, the call

```
ioctl(sfd, FIOCLEX, NULL)
```

will have the same effect. (See *fcntl(2)* and *ioctl(2)* for more information.)

Be Sure a Restricted Root Really Restricts

The *chroot(2)* system call, which may be used only by *root*, will force the process to

treat the argument directory as the root of the file system. For example, the call

```
chroot("/usr/riacs")
```

makes the root directory */usr/riacs* so far as the process which executed the system call is concerned. Further, the entry ‘..’ in the new root directory is interpreted as naming the root directory. Where symbolic links are available, they too are handled correctly.

However, it is possible for *root* to link directories just as an ordinary user links files. This is not done often, because it creates loops in the UNIX file system (and that creates problems for many programs), but it does occasionally occur. These directory links can be followed regardless of whether they remain in the subtree with the restricted root. To continue the example above, if */usr/demo* were linked to */usr/riacs/demo*, the sequence of commands

```
cd /demo  
cd ..
```

would make the current working directory be */usr*. Using relative path names at this point (since an initial ‘/’ is interpreted as */usr/riacs*), the user could access any file on the system. Therefore, when using this call, one must be certain that no directories are linked to any of the descendants of the new root.

Check the Environment In Which the Process Will Run

The environment to a large degree depends upon certain variables which are inherited from the parent process. Among these are the variables *PATH* (which controls the order and names of directories searched by the shell for programs to be executed), *IFS* (a list of characters which are treated as word separators), and the parent's *umask*, which controls the protection mode of files that the subprocess creates.

One of the more insidious threats comes from routines which rely on the shell to execute a program. (The routines to be wary of here are *popen*, *system*, *execvp(3)*, and *execvp(3)*.†) The danger is that the shell will not execute the program intended. As an example, suppose a program that is setuid to *root* uses *popen* to execute the program

† *execvp* and *execvp* are in section 2 of the System V manual.

printfile. As *popen* uses the shell to execute the command, all a user needs to do is to alter his PATH environment variable so that a private directory is checked before the system directories. Then, he writes his own program called *printfile* and puts it in that private directory. This private copy can do anything he likes. When the *popen* routine is executed, his private copy of *printfile* will be run, with *root* privileges.

On first blush, limiting the path to a known, safe path would seem to fix the problem. Alas, it does not. When the Bourne shell *sh* is used, there is an environment variable IFS which contains a list of characters that are to be treated as word separators. For example, if IFS is set to ‘o’, then the shell command *show* (which prints mail messages on the screen) will be treated as the command *sh* with one argument *w* (since the ‘o’ is treated as a blank). Hence, one could force the setuid process to execute a program other than the one intended.

Within a setuid program, all subprograms should be invoked by their full path name, or some path known to be safe should be prefixed to the command; and the IFS variable should be explicitly set to the empty string (which makes white space the only command separators).

The danger from a badly-set *umask* is that a world-writable file owned by the effective UID of a setuid process may be produced. When a setuid process must write to a file owned by the person who is running the setuid program, and that file must not be writable by anyone else, a subtle but nonetheless dangerous situation arises. The usual implementation is for the process to create the file, *chown*(2) it to the real UID and real GID of the process, and then write to it. However, if the *umask* is set to 0, and the process is interrupted after the file is created but before it is *chown*'ed the process will leave a world-writable file owned by the user who has the effective UID of the setuid process.

There are two ways to prevent this; the first is fairly simple, but requires the effective UID to be that of *root*. (The other method does not suffer from this restriction; it is described in a later section.) The *umask(2)* system call can be used to reset the *umask* within the setuid process so that the file is at

no time world-writable; this setting overrides any other, previous settings. Hence, simply reset *umask* to the desired value (such as 022, which prevents the file from being opened for writing by anyone other than the owner) and then open the file. (The *umask* can be reset afterwards without affecting the mode of the opened file.) Upon return, the process can safely *chown* the file to the real UID and GID of the process. (Incidentally, only *root* can *chown* a file, which is why this method will not work for programs the effective UID of which is not *root*.) Note that if the process is interrupted between the *open(2)* and the *chown* the resulting file will have the same UID and GID as the process' effective UID and GID, but the person who ran the process will not be able to write to that file (unless, of course, his UID and GID are the same as the process' effective UID and GID).

As a related problem, *umask* is often set to a dangerous value by the parent process; for example, if a daemon is started at boot time (from the file */etc/rc* or */etc/rc.local*), its default *umask* will be 0. Hence, any files it creates will be created world-writable unless the protection mask used in the system call creating the file is set otherwise.

Programming Style

Although threats from the environment create a number of security holes, inappropriate programming style creates many more. While many of the problems in programming style are fairly typical (see [4] for a discussion of programming style in general), some are unique to UNIX and some to setuid programs.

Do Not Write Interpreted Scripts That Are Setuid

Some versions of UNIX allow command scripts, such as shell scripts, to be made setuid. This is done by applying the setuid bit to the command interpreter used, and then interpreting the commands in the script. Unfortunately, given the power and complexity of many command interpreters, it is often possible to force them to perform actions which were not intended, and which allow the user to violate system security. This leaves the owner of the setuid script open to a devastating attack. In general, such scripts should be avoided.

As an example, suppose a site has a setuid script of *sh* commands. An attacker simply executes the script in such a way that the shell which interprets the commands appears to have been invoked by a person logging in. UNIX applies the setuid bit on the script to the shell, and since it appears to be a login shell, it becomes interactive. At that point, the attacker can type his own commands, regardless of what is in the script.

One way to avoid having a setuid script is to turn off the setuid bit on the script, and rather than calling the script directly, use a setuid program to invoke it. This program should take care to call the command interpreter by its full path name, and reset environment information such as file descriptors and environment variables to a known state. However, this method should be used only as a last resort and as a temporary measure, since with many command interpreters it is possible even under these conditions to force them to take some undesirable action.

Do Not Use creat for Locking

According to its manual page, “The mode given [*creat(2)*] is arbitrary; it need not allow writing. This feature has been used ... by programs to construct a simple exclusive locking mechanism.” In other words, one way to make a lock file is to *creat* a file with an unwritable mode (mode 000 is the most popular for this). Then, if another user tried to *creat* the same file, *creat* would fail, returning -1.

The only problem is that such a scheme does not work when at least one of the processes has *root*’s UID, because protection modes are ignored when the effective UID is that of *root*. Hence, *root* can overwrite the existing file regardless of its protection mode. To do locking in a setuid program, it is best to use *link(2)*. If a link to an already-existing file is attempted, *link* fails, even if the process doing the linking is a *root* process and the file is not owned by *root*.

With 4.2 Berkeley UNIX, an alternative is to use the *flock(3)* system call, but this has disadvantages (specifically, it creates advisory locks only, and it is not portable to other versions of UNIX).

The issue of covert channels [5] also arises here; that is, information can be sent illicitly

by controlling resources. However, this problem is much broader than the scope of this paper, so we shall pass over it.

Catch All Signals

When a process created by running a setuid file dumps core, the core file has the same UID as the real UID of the process.[†] By setting *umask*’s properly, it is possible to obtain a world-writable file owned by someone else. To prevent this, setuid programs should catch all signals possible.

Some signals, such as **SIGKILL** (in System V and 4.2BSD) and **SIGSTOP** (in 4.2BSD), cannot be caught. Moreover, on some versions of UNIX, such as Version 7, there is an inherent race condition in signal handlers. When a signal is caught, the signal trap is reset to its default value and *then* the handler is called. As a result, receiving the same signal immediately after a previous one will cause the signal to take effect regardless of whether it is being trapped. On such a version of UNIX, signals cannot be safely caught. However, if a signal is being *ignored*, sending the process a signal will *not* cause the default action to be reinstated; so, signals can be safely ignored.

The signals **SIGCHLD**, **SIGCONT**, **SIGTSTP**, **SIGTTIN**, and **SIGTTOU**[‡] (all of which relate to the stopping and starting of jobs and the termination of child processes) should be caught unless there is a specific reason not to do this, because if data is kept in a world-writable file, or data or lock files in a world-writable directory such as */tmp*, one can easily change information the process (presumably) relies upon. Note, however, that if a system call which creates a child (such as *system*, *popen*, or *fork(2)*) is used, the **SIGCHLD** signal will be sent to the process when the command given *system* is finished; in this case, it would be wise to ignore **SIGCHLD**.

[†] On some versions of UNIX, such as 4.2BSD, no core file is produced if the real and effective UIDs of the process differ.

[‡] The latter four are used by various versions of Berkeley UNIX and their derivatives to suspend and continue jobs. They do not exist on many UNIXes, including System V.

This brings us to our next point.

Be Sure Verification Really Verifies

When writing a setuid program, it is often tempting to assume data upon which decisions are based is reliable. For example, consider a spooler. One setuid process spools jobs, and another (called the *daemon*) runs them. Assuming that the spooled files were placed there by the spooler, and hence are "safe," is again a recipe for disaster; the *at* spooler discussed earlier is an example of this. Rather, the daemon should attempt to verify that the spooler placed the file there; for example, the spooler should log that a file was spooled, who spooled it, when it was spooled, and any other useful information, in a protected file, and the daemon should check the information in the log against the spooled file's attributes. With the problem involving *at*, since the log file is protected, the daemon would never execute a file not placed in the spool area by the spooler.

Make Only Safe Assumptions About Recovery Of Errors

If the setuid program encounters an unexpected situation that the program cannot handle (such as running out of file descriptors), the program should not attempt to correct for the situation. It should stop. This is the opposite of the standard programming maxim about robustness of programs, but there is a very good reason for this rule. When a program tries to handle an unknown or unexpected situation, very often the programmer has made certain assumptions which do not hold up; for example, early versions of the command *su*(1) made the assumption that if the password file could not be opened, something was disastrously wrong with the system and the person should be given *root* privileges so that he could fix the problem. Such assumptions can pose extreme danger to the system and its users.

When writing a setuid program, keep track of things that can go wrong – a command too long, an input line too long, data in the wrong format, a failed system call, and so forth – and at each step ask, "if this occurred, what should be done?" If none of the assumptions can be verified, or the assumptions do not cover all cases, at that point the setuid program should *stop*. Do not attempt to recover unless

recovery is guaranteed; it is too easy to produce undesirable side-effects in the process.

Once again, when writing a setuid program, if you are not sure how to handle a condition, *exit*. That way, the user cannot do any damage as a result of encountering (or creating) the condition.

For an excellent discussion of error detection and recovery under UNIX, see [6].

Be Careful With I/O Operations

When a setuid process must create and write to a file owned by the person who is running the setuid program, either of two problems may arise. If the setuid process does not have permission to create a file in the current working directory, the file cannot be created. Worse, it is possible that the file may be created and left writable by anyone. The usual implementation is for the process to create the file, *chown* it to the real UID and real GID of the process, and then write to it. However, if the *umask* is set to 0, and the process is interrupted after the file is created but before it is *chown*'ed, the process will leave a world-writable file owned by the user who has the effective UID of the setuid process.

The section on checking the environment described a method of dealing with this situation when the program is setuid to *root*. That method does not work when the program is setuid to some other user. In that case, the way to prevent a setuid program from creating a world-writable file owned by the effective UID of the process is far more complex, but eliminates the need for any *chown* system calls. In this method, the process *fork*(2)'s, and the child resets its effective UID and GID to the real UID and GID. The parent then writes the data to the child via *pipe*(2) rather than to the file; meanwhile, the child creates the file and copies the data from the pipe to the file. That way, the file is never owned by the user whose UID is the effective UID of the setuid process.

Some UNIX systems, notably 4.2BSD, allow a third method. The basic problem here is that the system call *setuid*(3)[†] can only set the effective UID to the real UID (unless the process runs with *root* privileges, in which

[†] This system call is in section 2 of the System V manual.

case both the effective and real UIDs are reset to the argument). Once the effective UID is reset with this call, the old effective UID can never be restored (again, unless the process runs with *root* privileges). So it is necessary to avoid resetting any UIDs when creating the file; this leads to the creation of another process or the use of *chown*. However, 4.2BSD provides the capability to reset the effective UID independently of the real UID using the system call *setreuid(2)*. A similar call, *setregid(2)*, exists for the real and effective GIDs. So, all the program need do is use these calls to exchange the effective and real UIDs, and the effective and real GIDs. That way, the old effective UID can be easily restored, and there will not be a problem creating a file owned by the person executing the setuid program.

Conclusion

To summarize, the rules to remember when writing a setuid program are:

- Be as restrictive as possible in choosing the UID and GID.
- Reset effective UIDs and GIDs before calling *exec*.
- Close all but necessary file descriptors before calling *exec*.
- Be sure a restricted root really restricts.
- Check the environment in which the process will run.
- Do not write interpreted scripts that are setuid.
- Do not use *creat* for locking.
- Catch all signals.
- Be sure verification really verifies.
- Make only safe assumptions about recovery of errors.
- Be careful with I/O operations.

Setuid programs are a device to allow users to acquire new privileges for a limited amount of time. As such, they provide a means for overriding the protection scheme designed into UNIX. Unfortunately, given the way protection is handled in UNIX, it is the best solution possible; anything else would require users to share passwords widely, or the UNIX kernel to be rewritten to allow access

lists for files and processes. For these reasons, setuid programs need to be written to keep the protection system as potent as possible even when they evade certain aspects of it. Thus, the designers and implementors of setuid programs should take great care when writing them.

Acknowledgements

Thanks to Bob Brown, Peter Denning, George Gobel, Chris Kent, Rich Kulawiec, Dawn Maneval, and Kirk Smith, who reviewed an earlier draft of this paper, and made many constructive suggestions.

References

- [1] Aleph-Null, "Computer Recreations," *Software - Practise and Experience* 1(2) pp. 201-204 (April-June 1971).
- [2] *UNIX System V Release 2.0 Programmer Reference Manual*, DEC Processor Version, AT&T Technologies (April 1984).
- [3] *UNIX Programmer's Manual, 4.2 Berkeley Software Distribution, Virtual VAX-11 Version*, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA (August 1983).
- [4] Kernighan, Brian and Plauger, P., *The Elements of Programming Style, Second Edition*, McGraw-Hill Book Company, New York, NY (1978).
- [5] Lampson, Butler, "A Note on the Confinement Problem," *CACM* 16(10) pp. 613-615 (October 1973).
- [6] Darwin, Ian and Collyer, Geoff, "Can't Happen or /* NOTREACHED */ or Real Programs Dump Core," 1985 Winter USENIX Proceedings (January 1985).

:login:

An Overview of the Sprite Project

John Ousterhout

Andrew Cherenson

Fred Douglis

Michael Nelson

Brent Welch

Computer Science Division

Department of Electrical Engineering and Computer Sciences

University of California

Berkeley, CA 94720

sprinters@arpa.berkeley.edu

Introduction

Sprite is a new operating system that we have been designing and implementing at U.C. Berkeley over the last two years. The design was strongly influenced by three pieces of technology: local-area networks, large physical memories, and multiprocessor workstations. This article is a brief summary of how the Sprite design reflects those technologies.

The single most important issue for us is the network. We hope to provide simple and efficient mechanisms for a collection of workstations (nodes) to work together over a local-area network or small internet. Ideally, all of the resources of the collection of nodes (files, devices, CPU cycles, etc.) should be transparently and efficiently accessible by each of the nodes. We would like to achieve an effect something like timesharing in its ease of sharing, but with the high performance and guaranteed interactive response that personal workstations can provide. We'd like Sprite to support communities of a few dozen to a few hundred people; growth beyond this size is less important to us than the "quality of life" we provide for smaller communities. Sprite provides two network-oriented features: a network file system and a process migration facility.

The second piece of technology that influenced Sprite's design is the ever-increasing amount of physical memory available on workstations. Sprite takes advantage of large memories with large file caches, both on clients and servers. The nodes use a simple consistency protocol to coordinate shared access to files. The file caches adjust their sizes dynamically, trading memory back and forth with the virtual memory system.

The third technological influence on Sprite is multiprocessors, which appear likely to become available in workstation configurations within a few years. To support multiprocessors, Sprite implements shared address spaces and has a multi-threaded kernel.

Basics

There are several other experimental or commercial operating systems that share some of Sprite's goals, particularly those related to networks. These include LOCUS [6], V [2], Accent [3], and systems marketed by Sun and Apollo. Of these systems, Sprite is most like LOCUS and Sun's version of UNIX, in that the kernel-call interface provided to user programs is almost identical to that of 4.3 BSD UNIX. However, while the kernels used by LOCUS and Sun are based on early versions of the BSD kernel, Sprite's kernel has been rebuilt from scratch. The Sprite kernel differs from the BSD-derived kernels in two major ways: it is based on a general-purpose kernel-to-kernel remote procedure call package; and it is multi-threaded.

The most important feature of the Sprite kernel implementation is its remote-procedure-call (RPC) facility, which the kernels of different nodes use to request services of each other [7]. The lightweight connection/acknowledgement protocol in Sprite's RPC mechanism is similar to the scheme of Birrell and Nelson [1]. Each kernel contains a small set of server processes that respond to incoming requests (the exact number of processes varies dynamically in response to the machine's RPC load). We have not yet implemented any particular security mechanisms: each of the kernels trusts each of the other kernels. Since we did not have

access to a stub generator, the server and client stubs for each call were written by hand (there are a few dozen of these).

The basic round-trip time for a simple RPC with no parameters is about 5 ms on Sun-2 workstations. Unlike Birrell and Nelson's scheme, Sprite's RPC provides for fragmentation of large packets. This allows large blocks of data to be transferred between kernels at more than 300 kbytes/sec. on Sun-2 workstations, which is 2 to 3 times faster than we initially achieved without fragmentation. See [7] for a detailed description and performance analysis of Sprite's RPC mechanism.

Our second major kernel change was to support multiprocessors. The BSD kernels are single-threaded: a process executing in the kernel cannot be preempted and the system code depends on the fact that a process executing in the kernel has exclusive access to all of the kernel data structures. In a multiprocessor configuration, it is not safe for more than one process to be executing kernel code at a time. In Sprite, we used a monitor-like approach, with explicit locks on modules and data structures. This allows more than one process to execute kernel code at the same time, which will be important when multiprocessor workstations become available.

The Network File System

The network file system is the heart of Sprite. As in most timeshared UNIX systems, the Sprite file system provides a shared mechanism for naming, for long-term storage, and for interprocess communication. To a network of Sprite workstations, there appears to be a single hierarchical UNIX-like file system with a single shared root. The same name yields the same file on every node of the system. The basic operations on Sprite files (open, close, read, write, seek, ioctl, lock, unlock, etc.) behave the same as if they were executed on a single BSD timesharing system, instead of a network of workstations. This differs from Sun's NFS, which does not support operations requiring state to be maintained on the server (e.g. lock and unlock).

The three most unusual aspects of the Sprite file system are its use of *prefix tables* for server location, its implementation of client-level caching, and its named pipes.

In any network file system with multiple servers, there must be a mechanism to determine which server is responsible for which files. Typically, each server handles one or more subtrees of the file hierarchy. In the LOCUS and NFS file systems, each client maintains a static *mount table* that associates servers with subtrees. There are no automatic mechanisms to update mount tables when server configurations change; as a consequence, large network systems are difficult to administer. In contrast, Sprite uses a more dynamic form of table called a *prefix table*. Each client maintains a small table that associates servers with subtrees using prefixes. When opening a file, the first few characters of the file's name are matched against each of the prefixes in the table; the open request is then sent to the server associated with the longest matching prefix. The prefix tables are created and modified dynamically using a simple broadcast protocol. There are no static global tables identifying the system configuration; all that is needed is for each server to know which prefixes it serves and to respond to broadcasts for information about those prefixes. Prefix tables are also flexible enough to handle private local subtrees and even a simple form of replication; see [8] for details.

In Sprite, both clients and servers keep main-memory caches of recently-used disk blocks. As memories get larger and larger, we expect these caches to get larger and larger too. Within a few years, cache hit ratios of nearly 100% should be common (see [5] for trace-driven predictions of cache effectiveness). File-system block caches have two advantages in a network operating system. First, they reduce disk traffic; our preliminary measurements suggest that this alone can result in factors of 2 to 5 in file-system throughput. Second, they eliminate network traffic between clients and servers, which provides an additional performance improvement.

The main problem with client file caches is cache consistency. If multiple clients are reading and writing a file at the same time, each read request must return the most recent information written to the file, regardless of which nodes are making read and write requests. Sprite guarantees cache consistency by disabling client caching for any file that is open for writing on one node while also open for reading or writing on some other node.

Consistency is implemented by the servers, with consistency actions being taken when files are opened or closed. Each server keeps track of which of its files are open on which nodes; when a server receives an open request that will result in conflicting access to a file, client caching (for that one file) is disabled on all nodes and the file's dirty blocks (if any) are flushed back from client caches to the server. This scheme permits caching under multiple-reader access, and also allows unshared files with short lifetimes to be created, used, and deleted entirely within a client cache, without ever being written to the server.

Although Sprite's approach suffers a performance degradation when consistency conflicts cause client caching to be disabled for a file, our measurements of current UNIX systems indicate that files tend to be open only for very short intervals (reducing the likelihood of a conflict), and that write-sharing of files is extremely infrequent [5]. An alternate approach is that of LOCUS, which is based on token-passing. One node (the current owner of the file's token) is permitted to cache blocks of a file, even when the file is open by multiple readers and writers. The other nodes must wait until they receive the token before accessing the file. In comparison to Sprite's mechanism, the token-passing approach requires extra implementation complexity and seems unlikely to reduce server traffic, since client caches must be flushed whenever the token changes hands.

In addition to the standard UNIX file facilities, Sprite provides *named pipes*, which combine the features of traditional UNIX files and pipes. We expect named pipes to be used instead of TCP/IP for inter-process communication within a Sprite system. A named pipe has a name in the file system, occupies space on disk, and persists even when not open by any processes. However, it has the operations of a normal pipe. Reads return the oldest data in the pipe and remove that data from the pipe atomically. Writes atomically append data to the end of the pipe. Processes attempting to read from an empty named pipe are suspended until data is written to the pipe. Sprite named pipes are similar to LOCUS named pipes, except that a Sprite named pipe retains its information even when not open (in LOCUS, un-opened named pipes are always empty). Client and server caches are used to

avoid writing pipe information to disk unless it is long-lived; we expect this to result in performance at least as good as TCP/IP.

Virtual Memory

In comparison to the BSD implementation of virtual memory, there are two major changes in Sprite: shared address spaces and backing files.

In anticipation of multiprocessor workstations, and to support closely-cooperating teams of processes, Sprite provides a simple form of address-space sharing. As in traditional UNIX, each process's address space consists of three segments: read-only code, writable heap, and writable stack. In UNIX, however, only the code segment may be shared between processes. In Sprite, both code and heap may be shared. Stacks are still private. An argument to the *fork* system call determines whether or not the new process will share the heap of the old process. This mechanism provides all-or-nothing sharing. It is suitable for a collection of processes all working on the same application, but does not allow a process to share one area of memory with one set of processes and a different area with a different set of processes. The Sprite scheme is simpler than the region-based memory mechanisms being discussed for 4.4 BSD, but less powerful.

UNIX has traditionally used a special area of disk for paging storage, with special-purpose algorithms for managing the paging storage. In contrast, Sprite uses ordinary files for paging storage. Each of a process's three segments has a corresponding backing file. For the code, it is the binary file from which the program was loaded (unless the code is to be writable); for the stack and heap, temporary files are allocated in a special directory of the file system. Ordinary file operations are used to read and write the backing files; storage for the backing files is not allocated until pages are actually written out [4].

Sprite's use of backing files has several advantages over the traditional UNIX approach. First, it is simpler, since no special algorithms need to be implemented for managing backing store. Second, by using the network file system for backing files, the paging information can easily be accessed by all the nodes on the network; this is important for process migration (see below). Third, it allows

the server's disk cache to serve as an extended memory for the clients it serves. Fourth, it saves disk space since backing storage is only allocated when needed. In contrast, the BSD-based implementations of virtual memory allocate statically a separate area of disk for each workstation to use for backing storage. The result is that backing storage occupies a huge amount of disk space on the file servers, with very little of it in actual use at any given time. With the large block sizes and clever disk allocation mechanisms used by modern file systems, we don't see any performance advantage to using a special-purpose paging store.

Virtual Memory vs. File-System Cache

Both the virtual-memory system and the file cache require physical memory, and each would like to use as much memory as possible. In traditional UNIX, the file cache is fixed in size at system boot time, and the virtual-memory system uses what's left. For Sprite, we decided to permit the boundary between virtual memory and file system to change dynamically. When not much memory is needed for virtual memory, the file cache can grow to fill the unused memory; when more memory is needed for VM, the file cache shrinks. This allows applications with small virtual memory needs (such as the compiler and editor) to use almost all of memory as a large file cache, while permitting larger applications to use all of memory for the pages of their address spaces.

To implement the flexible boundary between the file cache and VM, each component manages its pages using an LRU-like algorithm (perfect LRU for the file cache, clock for VM). When VM needs a new page, it compares the age of its oldest page with the age of the file cache's oldest page. If the file cache's page is older, then the file cache frees up a page and gives it to VM. A similar approach is used by the file cache with it needs a new page.

Process Migration

Sprite allows processes to be migrated between nodes with compatible instruction sets. Process migration is simplified by the availability of a shared network file system (which allows the process to continue to access the same files from its new node), and the use

of files for backing store. To migrate a process from node A to node B, node A writes out all of the process's dirty pages to the file server and transfers the execution state to node B. Node B then reloads the process's pages from the file server on a demand basis as the process executes. The file-system cache-consistency protocols guarantee that any file modifications made by the process on node A will also be visible to the process on node B.

The cost of process migration is determined primarily by the number of dirty pages in its address space. On Sun-2 workstations, an empty process can be migrated in about 0.5 second, with dirty pages flushed at about 100 kbytes/sec.

Although many system calls (e.g. those related to the file system) are already location-independent, there are others that may have different effects if executed on different machines (e.g. "get time of day"). To handle the non-transparent system calls, Sprite assigns to each process a *home node*; location-sensitive kernel calls are sent to the home node using the kernel-to-kernel RPC mechanism, and are processed on the home node. This guarantees that a process produces exactly the same results whether or not it has been migrated. Although sending system calls home is expensive, our preliminary measurements indicate that this happens infrequently, and that migrated processes suffer no more than a 5-10% performance degradation.

Although the basic process migration mechanism is now running, we haven't yet implemented the policy mechanisms to accompany it. We plan to provide facilities akin to the '&', 'bg', and 'fg' facilities of *csh*, which will offload a process to an idle workstation instead of putting it in background. Sprite will keep track of which workstations are idle (e.g. those whose users have not been active for at least 10 minutes) and choose one of them for the offloading. If a user returns to a workstation on which processes have been offloaded, the foreign processes will be ejected, either back to their home node or to another idle node. We are also implementing a new version of the *make* utility called *pmake*, which uses the process migration facilities to rebuild large systems in parallel.

Project Status

The Sprite team was formed in the fall of 1984, and we began coding in the spring of 1985. As of today, all of the major pieces of the system are operational except for the site-selection mechanisms of process migration and some recovery code. We are currently in the phase of the project where bugs are appearing at least as fast as we can fix them, but we hope that the system will stabilize enough for us to start using it for everyday work sometime in the next several months. We expect to have detailed performance measurements available within a few months. At this point all we can say is that our initial measurements on the untuned system are very encouraging (aren't they always?). It's still too early to make any conclusions about the success of our design decisions.

Other Operating-System Work at Berkeley

Sprite is not the only operating system project underway at Berkeley. Two others that you may also be interested in are the DASH project and the 4.n BSD work. DASH is a project being led by Profs. David Anderson and Domenico Ferrari. The letters of the name stand for "Distributed," "Autonomous," "Secure," and "Heterogenous"; the DASH project is exploring these issues in the context of very large internetworks (whereas Sprite is concerned primarily with tightly-cooperating, mutually-trusting, homogeneous groups of machines that are not widely-distributed). The 4.n BSD work, in which Mike Karels and Kirk McKusick are the principals, should need no further introduction to this community; it is continuing the evolution of Berkeley UNIX with new features such as region-based memory management and a more flexible file system structure to permit multiple network file systems to co-exist.

References

- [1] Birrell, A. D. and Nelson, B. J. "Implementing Remote Procedure Calls." *ACM Transactions on Computer Systems*, Vol. 2, No. 1, February 1984, pp. 39-59.
- [2] Cheriton, D. R. "The V Kernel: A Software Base for Distributed Systems." *IEEE Software*, Vol. 1, No. 2, April 1984, pp. 19-42.
- [3] Fitzgerald, R. and Rashid, R. F. "The Integration of Virtual Memory Management and Interprocess Communication in Accent." *ACM Transactions on Computer Systems*, Vol. 4, No. 2, May 1986, pp. 147-177.
- [4] Nelson, M. *Virtual Memory for the Sprite Operating System*, Technical Report UCB/CSD 86/301, Computer Science Division, University of California, Berkeley, June 1986.
- [5] Ousterhout, J. K. et al. "A Trace-Driven Analysis of the 4.2 BSD UNIX File System." Proceedings of the 10th Symposium on Operating Systems Principles, *Operating Systems Review*, Vol. 19, No. 5, December 1985, pp. 15-24.
- [6] Walker, B. et al. "The LOCUS Distributed Operating System." Proceedings of the 9th Symposium on Operating Systems Principles, *Operating Systems Review*, Vol. 17, No. 5, November 1983, pp. 49-70.
- [7] Welch, B. *The Sprite Remote Procedure Call System*, Technical Report UCB/CSD 86/302, Computer Science Division, University of California, Berkeley, June 1986.
- [8] Welch, B. and Ousterhout, J. "Prefix Tables: A Simple Mechanism for Locating Files in a Distributed System." *Proceedings of the 6th International Conference on Distributed Computing Systems*, May 1986, pp. 184-189.

;login:

Book Review

The C Programmer's Handbook by M. I. Bolsky (AT&T Systems Center)
(New York: Prentice-Hall, Inc., 1985) \$16.95

Reviewed by Marc D. Donner

IBM Thomas J. Watson Research Center
ucbvax!ibm.com!donner

Quick! What is the order of arguments of *ungetc*? Is it *ungetc(char, stream)* or is it *ungetc(stream, char)*? Well, you grab your copy of K&R, go to the index. Hmm, let's see, that's pretty far down. Ah, here it is in the index on page 228. Now you thumb through the book looking for page 156 ... and here it is near the bottom ... *ungetc(c, fp)*.[†]

On the other hand, I grab my copy of *The C Programmer's Handbook* and turn it over and look at the index to library functions printed on the back cover. It tells me that *ungetc* is documented on page 49. I flip the book open to page 49, quite easy because the numbers are printed in a large font at the outside upper corner of each page, the pages are thick, and there aren't too many in the handbook. And there it is, at the bottom, *ungetc(c, stream)*. And it tells me that *c* is an *int* and that *stream* is a *FILE**!

There are many unusual things about this handbook, and all of them contribute to making it one of the most usable books I possess. On the front cover is printed the table of contents. The back cover has the index to library functions plus a list of all of the operators in precedence order, with notations about whether they associate right-to-left or left-to-right. The book is spiral bound so that it lies flat, a big plus when you want to refer to something while hacking. How much would you pay for a spiral bound copy of K&R? The pages are uncrowded, making the information more easily locatable. The format is simple and consistent throughout, again contributing to ease of use. The paper is unusual ... it is fairly thick and of high quality, making the resulting pages easy to handle. Normally, thick paper is low quality, but this book seems to be an exception.

[†] By the way, I asked this question of a prominent UNIX guru of my acquaintance while writing this review and he proposed two answers, both wrong; so this is not a trivial question.

Each entry in the book is introduced by a red headline. Structurally each entry is an unordered list, generally containing some syntax, some explanation, an example when appropriate, and a reference to the section in the on-line manual pages where more detail resides. The designers weren't intimidated by white space, and they use it to considerable advantage, making the entries easy to read and find.

One minor drawback is the choice of sanserif fonts for the printing. It is well known that serifed fonts are more readable, though sanserif fonts are widely believed to be more "modern." Another, more serious problem, is the fact that this handbook is specific to System V. I have been using it while programming my 4.2 system for several months now and the only mismatch that has really bothered me is that the 4.2 *index* function is called *strchr* in the System V library. I find the name *index* to be more informative, though I can certainly understand the choice of *strchr*. In an ideal world, the book would be available in a 4.2 version, also.

I suppose that reviewing a book like this is like reviewing the telephone book ... a bit thin on plot, but interesting typesetting. There is almost no writing in this book, it is the closest thing to a real handbook that I have ever seen going under that name. This book is deceptively simple, but a great deal of thought must have gone into each decision. Even such choices as the paper it is printed on seem to have received careful attention.

Please don't be overwhelmed by my enthusiasm, this book is not the be-all and end-all, but it is very nice to see something simple like this done well once in a while. Something that doesn't pretend to be much, but is a most excellent example of what it is.

:login:

Standards

Ima Tired

You know what I think? I think that people are not taking this standards thing very seriously.

Standards are important things, you know. Ignoring Gregorian chants, Ben Franklin was among the first to use standards. He recommended the use of interchangeable parts in rifles. This reduced their downtime, increased their performance, and considerably increased their repairability.

Imagine living in the olden times and breaking a part on your rifle:

He: Hey Rachel, the flintlock's broken.
She: Bad news, Harry. Our gunsmith, Withers, passed away last year. You're going to have to get a whole new gun.
He: You mean no one has a new flintlock?
She: Nope, they're one-of-a-kind. Only Withers knew how ours worked.

Ben had many good ideas. Henry Ford carried them to perfection.

So where are we now? There are standards everywhere, you know. Every time you look at a screw, a nail, a brick, a board ... all are manufactured to standard sizes. (NB: Some standards are "soft," e.g., the 2.54 cm nail standard.)

Computer manufacturers kind of have standards. Consider characters. I learned on a Bendix G-15. It had the extended character set option and could actually print letters instead of just numbers. This was a startling innovation (all the more startling due to its blazing speed – three characters/second!). The G-15 had 29-bit words and bizarre encoding for the characters. The coding scheme died a merciful death.

By the 60's, the ASCII code emerged. You know: the American Standard Code for Information Interchange. All American manufacturers were to adhere to it voluntarily. Every one of them. Except CDC, which was busy with 6-bit character codes, 10 per word. Except PLATO, CDC machines which had variable length characters (6 to 24 bits). Except UNIVAC; they used "field data," more 6-bit characters. Not too many special

characters there, nosirree! Case distinctions? Who needs it! They also had the "quarter-word" format which stored four 9-bit characters. That was enough for upper/lower case and some exciting nonstandard graphics. Except DEC. Their DECsystem-10 had a scheme which encoded characters using a MOD-50 scheme. Innovative.

And: except IBM. They decided to use EBCDIC instead. Terrific. Instead of the ISO or any other standard, they had a new kind: the de facto standard. The phrase "de facto" means "everyone does it this way so it doesn't matter what you think." IBM is real big on de facto standards.

Time passed; the world turned around once every day. Soon people found that the fewer ways there were to do a given thing (e.g., character codes) the more productive they could be. The world of computers has seen many standards emerge in recent years. Early on, tape formats standardized, thus enhancing interchange of data among various systems. Local area networks fueled the need for standards as each manufacturer found they needed to meet some level of compatibility or die. The personal computer world has almost achieved the world of plug-and-play for some kinds of peripherals and computers. What an amazing universe we now live in.

So what's the complaint? I'll tell you the complaint: the very word "standard" is now bantied about as if it means "latest way we invented to do something." When's the last time YOU said, "Oh, I think I'll invent a new page-description language; we can make it a standard!" Or maybe: "Gosh, I don't think there's enough network file systems in the world; let's have a NEW STANDARD." AT&T tried that one. Oops.

Standards are hard. Either you have to let one guy (maybe two if they're friends) do it or you have to have a "committee." Committees can do it: it's been proven. Unfortunately, they take longer. The last FORTRAN standard took 10 years. The next one, currently dubbed FORTRAN 8x may not make it! It may turn out to be FORTRAN 9x. How disappointing.

. . . continued

:login:

At any rate, while companies like IBM can create de facto standards just because they sell some substantial fraction of every computer in the world, that doesn't mean just anyone can.

Let's all see how we can cooperate in the coming year and have just a few standards – a few good ones.

Letters to the Editor

Date: Wed, 14 Jan 87 20:17:53 AEST
From: robf@runx (Rob Fowler)
To: auugn@monu1
Subject: Possible suggestion for newsletter content.

I noticed that the newsletter got a little short this time round. Too bad.

I would like to write something for it but I am afraid that my knowledge is insignificant compared to some of the heavies that read it.

Maybe you could have some more technical details of the newer stuff like V8, or are most of the other readers in a position where that is not of interest to them? I would particularly like to see some details and discussion on V8 streams. Would the newsletter be the wrong place for such an article (eg copywrite considerations?) I know that most of the technical stuff that can be discussed is already discussed on the net.

I am not an academic so I do not have access to a lot of the leading edge 'super toys' eg the BLIT, but I do enjoy reading about them. (Only have PCAT's with XENIX V to play/work with).

I REALLY liked Volume 7 Number 1, I read every word with interest.

Rob Fowler.

I hope someone will help out here by sending an article on one of the subjects that Rob would like to appear in AUUGN.

AUUGN Editor.

O'REILLY & ASSOCIATES, INC.

S P E C I A L I S T S I N T E C H N I C A L C O M M U N I C A T I O N

November 25, 1986

John Carey
AUUG Editor
Computer Center
Monash University
Clayton, Victoria 3168
Australia

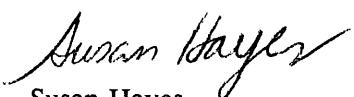
Dear Mr. Carey:

Thank you for your letter requesting review copies of the Nutshell Handbooks. We would be very happy to have you review them. I am enclosing the handbooks on *UNIX*, *vi*, *termcap*, and *curses*. Please let us know how you like them.

Incidentally, many universities have found our Nutshell Handbooks to be ideal training tools. They are inexpensive and to the point, and work well for both staff and classroom instruction.

Mr. Carey, have an enjoyable holiday season.

Sincerely yours,



Susan Hayes
Marketing

Please would someone volunteer to review these for the AUUGN.

Also any other publisher is welcome to send books for review.

AUUGN Editor

Softway Pty Limited
(Inc in NSW) PO Box 305
First Floor, 120 Chalmers Street
Strawberry Hills 2012 NSW Australia
☎ (02) 698 2322 Fax (02) 957 6914



Mr John Carey
Editor - AUUGN
Computer Centre
Monash University
Clayton VIC 3168

Tuesday 27th January, 1987

Dear Sir,

I would like to protest about the recent advertisement for AT&T Unix Pacific on pages 38 and 40 inclusive in AUUGN Volume 7 Number 2-3.

Here is an enormous multi-national organisation who is allowed *free* press in our struggling but Australian owned informative newsletter. Afterall, AT&T as a whole, outnumber us at the rate of 1000:1. They are going to make some Australian dollars out of the sale of this newsletter, in fact, about twice as many dollars as AUUG.

Since we have agreed at the beginning of our newsletter, that advertisements were to be charged at the rate \$200.00 per page, I think that we certainly would have been justified in asking for a donation at least. I am well aware there is also a mention about "items of general interest" being given space in the newsletter.

Please find enclosed an advertisement for our company with a cheque for \$200.00.

Yours sincerely

Greg Rose,
Managing Director.



for

UNIX System V

Documentor's Workbench 2.0

- and various back-end drivers
- PostScript support of plain text
- support for graphs and images

Ports & Device Drivers

Intelligent Benchmarking

SUN-III (ACSnet) + installation

**Biway - Bi-directional modem software for System V
and 4bsd**

Courses:

- Beginner's Workshop
- Fast start to UNIX
- System Administrators' workshop

Technical Backup

- and all sorts of interesting software development.

Softway Pty Ltd. (Incorporated in NSW)
120 Chalmers St, Strawberry Hills, NSW.
PO Box 305, Strawberry Hills, NSW 2012.
 (02) 698 2322 Fax (02) 957 6914

Computer Centre
Monash University
Clayton, Victoria 3168

Wednesday 18th February, 1987

Greg Rose
Managing Director
Softway Pty. Ltd.
First Floor
120 Chalmers Street
Strawberry Hills
N.S.W. 2012

Dear Greg,

Thank you for your letter dated the 27th of January 1987. I note your concern that such a *small* organisation as the AUUG is providing AT&T with free advertising in its Newsletter.

I do feel that the information that is sent to me by AT&T is of *general interest* to the AUUGN readership. This is because AT&T announcements could affect in some way the environment in which we all work and should be published as a service to the AUUG members.

You may disagree !!

To your probable chagrin, I have included in the current issue of the AUUGN the AT&T press releases which were presented at UNIFORUM in January this year.

Perhaps AT&T could help with the production of the AUUGN in return for the goodwill it would generate between the two organisations. This would not be unusual as USENIX receives support from several vendors in the production of their newsletter ;login:.

I have sent a copy of your letter and this reply, with a request for assistance to:-

Ryserson E. Schwark
Account Executive - Software Licensing
AT&T Unix Pacific

I wait eagerly for his reply.

Yours Sincerely,



John Carey,
AUUGN Editor.

P.S. A donation to the AUUGN of Steven Frede's excellent *ditroff* driver for the Apple LaserWriter

sold by Softway would not hurt either ; -)

P.P.S. Sorry, I don't have a happy face glyph.



AT&T Unix Pacific Co.,Ltd.
No. 1 Nan-oh Bldg., 5th Fl.
2-21-2, Nishi-Shinbashi
Minato-ku, Tokyo 105 Japan
Tel : 03-431-3305
Telefax : 03-431-3680
Telex : J34936 ATTUP

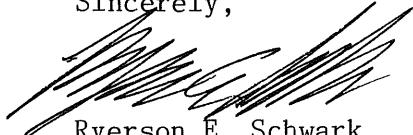
February 2, 1987

Mr. John Carey
AUUGN Editor
Computer Centre
Monash University
Clayton, Victoria 3168
Australia

Dear Mr. Carey,

I am enclosing press releases released by AT&T at UNIFORUM January 20-23. I hope that you will find the enclosed information useful. As you know, AT&T Unix Pacific is responsible for source code licensing of Unix* System V and related source code products in this region as well as doing development work. Should you have any questions about the enclosed material or other matters, please feel free to contact us.

Sincerely,



Ryerson E. Schwark
Account Executive
Software Licensing

Tokyo-Japan-RS-hy

* Registered Trademark of AT&T in the USA and other countries



AT&T PRESS RELEASE AT UNIFORUM IN WASHINGTON D. C.

JANUARY 20-23, 1987

SYSTEM V VERIFICATION SUITE, RELEASE 3

SYSTEM V INTERFACE DEFINITION, ISSUE 2, VOLUME 3

AT&T JOINS X/OPEN

AT&T SUPPORTS IEEE POSIX EFFORTS

SYSTEM V RELEASE 3.0 LICENSING SUCCESS

SYSTEM V RELEASE 3.0 MICROPROCESSOR PORTS

UNIX* SYSTEM V RELEASE 3 PRODUCT SUMMARY

* Registered trademark of AT&T in the USA and other countries.



AT&T Unix Pacific Co., Ltd.
No.1 Nan-oh Bldg., 5th Fl.
2-21-2, Nishi-Shinbashi
Minato-ku, Tokyo 105
Japan

For further information:

Ryerson Schwark
Software Licensing Account Executive
Tel: 3-431-3305 (Japanese)
3-431-3670 (English)
Fax: 3-431-3680
Telex: J34936 ATTUP
uucp:seismo!akgua!attunix!upshowa!schwark

AT&T Press Release -- UNIFORUM 1987

The role of UNIX* System V in AT&T's data networking strategy and the evolution of national and international standards for the operating system were reflected in several announcements made at the UNIFORUM conference and trade show January 20 - 23, 1987.

The announcements included updated editions of AT&T's SVVS (System V Verification Suite Release 3) and SVID (System V Interface Definition Issue 2, Volume III). AT&T, a member of the SIGMA Project in Japan, announced that it has become a full member of X/OPEN, an international industry consortium of hardware vendors in Europe.

Since UNIX System V Release 3 was announced in June 1986, AT&T's goals have been to accelerate the worldwide licensing of the system and to support the demands for worldwide standards for UNIX System V, according to Mike DeFazio, AT&T Director of Software Systems.

"The benefits of System V Release 3 to business are apparent in the worldwide demand for system licenses from computer system and chip vendors, OEM's, and software developers," said

* Registered trademark of AT&T in the USA and other countries.

DeFazio.

"Current UNIX System V Release 3 licensees shipped more than 75 percent of all UNIX system based products in 1985, so UNIX system industry leaders are obviously embracing the latest technology."

All major US chip manufacturers, including AT&T, Fairchild Semiconductor Corporation, Intel Corporation, Motorola Inc., National Semiconductor Corporation, and Zilog, have licensed UNIX System V Release 3.

DeFazio also stressed AT&T's support of the IEEE's 1003.1 standard for Portable Operating System for Computer Environments (POSIX). "The objectives of UNIX System V are fully compatible with the objectives of POSIX," he said. "We expect System V to conform to the future POSIX standard and we are confident that this will result in substantial savings for computer users and manufacturers."

DeFazio said that the installed base of UNIX computers today is about 60 times larger than it was when System V was first announced. He noted that the UNIX system installed base grew by about 75 percent in the past year alone.

AT&T demonstrated its Information Systems Network (ISN), STARLAN local area network, and ACCUNET®, an X.25 packet switching service which is available from AT&T in the US and internationally.

AT&T Unix Pacific demonstrated the second release of the Japanese Application Environment. This software was shown demonstrating the use of multiple languages on one AT&T 3B2 computer. Korean, Japanese, German and English were the multiple languages demonstrated.

Also exhibited were AT&T's PC 6300 and PC 6300 Plus, UNIX PC, and 3B2 minicomputer. Two recently announced data terminals, the 615 MT (Multitasking Terminal) and the 620 MTG (Multitasking Terminal with Graphics) were also demonstrated.

* * * *



System V Verification Suite (Release 3.0)

For Various Hardware Systems

Product Overview:

The System V Verification Suite (SVVS) is a set of test programs that verify that a port of the AT&T UNIX* System V Operating System conforms to the System V Interface Definition (SVID). SVVS may be used by computer system vendors, software vendors, or their customers to verify conformance to the SVID.

SVVS tests the operating system for conformance to the SVID Base Definition, which specifies an operating system environment where application software can be written independent of a particular hardware architecture. The presence of the SVID Base is required in all products derived from System V Release 3.0. The SVID Base Definition defines a basic set of System V components required by application programs, including each component's source code interface and runtime behavior. In addition, the system defines the Kernel Extension, which consists of the operating system services and utilities. The Network Services Extension consists of a new set of tests that verify the Transport Layer Interface, the Streams Interface, and Remote File Sharing. The Terminal Interface Extension tests the Terminal Interface facilities as described in Volume III of the SVID, Issue 2.

The SVVS Directory Structure contains all required directories and fits into one directory tree. The user interface provides a menu-driven capability with a choice of various testing options. Report and Verify commands inform the user of the SVVS conformity status. The Test Driver creates journals, cleans up temporary files created by the tests and controls dependencies and the order in which the tests are invoked.

SVVS tests are invoked through an easy-to-use, menu driven User Interface. A report generator formats the test results and creates a report which describes validation errors.

SVVS requires 30 Megabytes of disk storage to hold the largest section of directories and compiled binaries. SVVS can be installed on a section-by-section basis. SVVS is available only in source form.

Features:

- Tests all operating system services and general library routines listed in the SVID BASE and BASE Addendum.
- Tests the BASE, Kernel Extension (KEXT), Terminal Interface, and Network Services environments and special device files as defined in the SVID BASE and BASE Addendum.
- Tests the operating system service routines of the KEXT as listed in Volume I of the SVID, Issue 2.
- Tests the operating system library routines of the Terminal Interface Extension (TIE), as listed in Volume III of the SVID, Issue 2.
- Tests the operating system services and library routines of the Network Services Extension (NSE), as listed in Volume III of the SVID, Issue 2.

* Registered trademark of AT&T in the USA and other countries.

- Easy-to-use installation procedures permit software implementation in less than one day.
- Directory Structure contains all required directories and fits into one directory tree.
- Menu-driven user interface.
- Report and Verify commands inform the user of SVID conformance status.
- Test Driver creates journals, cleans up temporary files and controls dependencies and various sections that are tested.
- Automatic validation process requires minimum user interface.
- Automatic journalization of output identifies problems and initiates problem resolution.
- Validation at the System V Release 2.0 or Release 3.0 levels.
- Reduced time and processing costs by testing for a specific function or group of functions.

For additional information:

AT&T Unix Pacific Co., Ltd.
No.1 Nan-oh Bldg., 5th Fl.
2-21-2, Nishi-Shinbashi
Minato-ku, Tokyo 105
Japan

Tel: 3-431-3305 (Japanese)
3-431-3670 (English)
Fax: 3-431-3680
Telex: J34936 ATTUP
uucp: upshowa!attup



AT&T Offers Enhanced System V Interface Definition

Consistent With System V Release 3.0

AT&T is now offering a new, third volume of the System V Interface Definition (SVID), Issue 2, the current issue of the document that specifies common system interfaces for implementations of System V. These volumes describe the operating system facilities customers can expect on any computer that provides the System V Interface, the interface to those facilities for both application programs and end-users, and the run-time behavior.

The purpose of the SVID is to define a consistent set of program interfaces to encourage the development of portable applications software. End-users will benefit by having a wide range of computers with compatible operating systems and applications. Applications written to the SVID will be portable to any machine that conforms to the SVID. End-users will be able to choose the computer that best meets their needs and still have available the same applications, thereby preserving their investment in both software and training.

Issue 1 of the SVID was published in January 1985. It consisted of one volume that described the components in the System V Base and the Kernel Extension. Issue 2 was published in February 1986. It consisted of two volumes: Volume 1, updated for greater clarity and precision, and Volume 2 defining five additional extensions: the Base Utilities Extension, the Advanced Utilities Extension, the Administered Systems Extension, the Software Development Extension, and the Terminal Interface Extension. Issue 2 now consists of three volumes. A new volume, Volume 3, includes an enhanced Terminal Interface Extension and defines an additional extension: the Network Services Extension.

The Base System Definition and Kernel Extension define a run-time environment for application programs. The Base Utilities Extension defines the user-level interface to and the expected run-time behavior of basic tools for directory and file manipulation, text editing and shell programming. The Advanced Utilities Extension defines more advanced tools, multi-user and multi-system communication utilities, and printer job queuing facilities. The Administered Systems Extension defines tools for system administrators. The Software Development Extension defines tools for the software developer to write and maintain programs written in the C language. The Network Services Extension allows multi-system communication support of networking applications. The Network Services Extension defines (1) facilities that allow the sharing of resources (e.g., files and printers) transparently across a network; (2) facilities that provide a modular and uniform mechanism for implementing network services and communications independent of any particular protocols by providing standard interfaces to protocol modules and devices and by providing flexible ways of interconnecting modules; and (3) facilities that provide user programs the transport level services defined in the Open Systems Interconnection (OSI) Reference Model independent of any specific transport protocol. The Terminal Interface Extension defines facilities that allow application programs to perform terminal-handling functions in a way that is independent of the type of terminal that is actually in use.

The SVID is a valuable tool to customers interested in having a consistent UNIX* system computing environment available on a wide range of machines from many vendors. This

includes end-users and application developers (ISVs) producing C language applications whose source code must be portable from one System V environment to another.

The SVID defines the common computing environment for System V Release 2.0 and System V Release 3.0, and describes future directions for the operating system. Another announced product, the System V Verification Suite, can be used by OEMs and VARs to test their systems to see that they meet the System V Interface Definition.

As new functionality and new operating system features are developed, they will be described in new Extensions to the SVID. Future issues of the SVID will describe advanced standard interfaces to support networking applications, advanced user-interface tools for applications, and tools for internationalizing the system.

All three volumes of the SVID are currently available. Individual volumes are US\$65.00 each. The three-volume set is US\$165.00. The prices include air mail postage. Orders should be sent to:

AT&T Unix Pacific Co., Ltd.
Documentation Dept.
No.1 Nan-oh Bldg., 5th Fl.
2-21-2, Nishi-Shinbashi
Minato-ku, Tokyo 105
Japan

Tel: 3-431-3305 (Japanese)
3-431-3670 (English)
Fax: 3-431-3680
Telex: J34936 ATTUP
uucp: upshowa!attup

* Registered trademark of AT&T in the USA and other countries.



AT&T Expands Relationship with X/OPEN

AT&T today announced that it has become a member of the X/OPEN group, a consortium of European and American computer manufacturers.

The purpose of the group is to increase the volume of applications available for member machines, and to maximize the return on investments in software development, by ensuring portability of application programs.

AT&T has worked with X/OPEN since 1985 -- at that time the group consisted of only European companies -- with the goal of establishing a common applications environment based on the AT&T System V Interface Definition.

X/OPEN is also focusing attention on the evolution of public standards for systems software, including interfaces for databases, graphics, and languages.

"X/OPEN has expanded its membership to include companies outside Europe," said William T. O'Shea, AT&T executive director of information technologies. "We want to work with X/OPEN because our goals align on the broad software issues they are now addressing and we feel we can increase our participation in the group by becoming a full member."

European members of X/OPEN include CM Bull, France; International Computers Limited (ICL), England; Ericsson, Sweden; Nixdorf Computer AG and Siemens Aktiengesellschaft, Federal Republic of Germany; Olivetti, Italy; Philips International NV, The Netherlands.

The United States members are Digital Equipment Corporation, Unisys, Hewlett-Packard, Inc., and AT&T.



AT&T Supports POSIX

The computer industry is engaged in developing standards for the portable operating system environment. Conformance to the standards will make it possible for applications written in the C language to achieve a high degree of portability when utilized within a heterogeneous hardware environment. The rapid acceptance of UNIX* System V among both computer users and manufacturers is a testament to the magnitude of the reduction in costs achievable through utilization of these standards.

The standardization activities now underway within the IEEE 1003 committees to achieve the objectives furthered by a portable operating system environment are fully supported by AT&T. It is AT&T's intention to support the full-use version of the IEEE 1003.1 Standard, Portable Operating System for Computer Environments (POSIX), when it is approved. Since UNIX System V has been so widely accepted, and System V objectives align with those of POSIX, we expect System V to conform to the approved POSIX standard.

* Registered trademark of AT&T in the USA and other countries.



Success of UNIX* System Licensing

System V Release 3.0 has proven to be an extremely successful product in its first six months of availability. For example, System V Release 3.0 was licensed at a faster rate than System V Release 2.0 during their respective startup months. Furthermore, all major US chip vendors, including AT&T, Motorola, Intel, National Semiconductor, Fairchild, and Zilog, currently have sublicensing rights for System V Release 3.0. Considering that the current System V Release 3.0 licensees were responsible for >75% of all UNIX system shipments in 1985, it is clear that the UNIX market leaders are embracing System V Release 3.0. Half of System V Release 3.0 source licensees with sublicensing provisions have also already licensed SVVS.

The UNIX system industry is experiencing tremendous growth at a time when much of the computer systems industry is experiencing minimal growth. For example, the installed base of UNIX computers grew by about 75% during the past year.¹ Another indicator of the success of the UNIX systems industry is the fact that the installed base of UNIX computers has grown by a factor of about 60 since AT&T introduced System V in 1983.

* Registered trademark of AT&T in the USA and other countries.

1. One year period ending September 30, 1986.



UNIX* SYSTEM V Release 3.0 Ported To Microprocessors

UNIX System V Release 3.0, introduced earlier this year, has gained growing recognition as a solution to many business problems centering on the need to network computers.

One result of that recognition is the expanding list of equipment and software vendors developing products based on System V Release 3. AT&T has announced its own 32-bit chip, the WE 32100, with UNIX System V Release 3.0.

Several prominent companies, including four major silicon chip manufacturers -- Fairchild Semiconductor Corp., Intel Corporation, Motorola, Inc., and National Semiconductor Corporation -- have announced that they are developing products derived from this release.

The ports of these and other vendors will conform to AT&T's System V Interface Definition (SVID). SVID conformance assures that a customer can use any applications software regardless of the hardware on which it was developed. This source level portability of applications gives customers great freedom in selecting hardware.

One of the ports of UNIX System V Release 3.0 is to the Intel 80386 chip. The port includes support for the chip's virtual 8086 mode and for the 80386 family of floating-point coprocessors. It also supports the ISO transport model for the Remote File Sharing (RFS) implementation.

Motorola said that its SYSTEM V/68 Release 3 is the standard operating system for its 68020 family of computer products. The kernel of the Motorola system is combined with the RFS feature of System V Release 3.0 to allow different computers to share data, files, applications and peripherals without specialized utilities.

National Semiconductor's Series 32000 microprocessors include RFS as well as Streams which provides for the development of communications services within System V Release 3.0.

Originally Released in November 1986

* Registered trademark of AT&T in the USA and other countries.



Product Summary

AT&T's UNIX* System V, Release 3.0

This Product Summary describes UNIX System V, Release 3.0, which has been available since June 30, 1986. This product introduces several technical innovations which, taken together, provide a powerful environment for developing, executing and supporting network applications.

UNIX System V, Release 3.0 includes the following major features:

Remote File Sharing (RFS):

The RFS feature provides the ability to share resources transparently across a network. Both files and devices (printers, tape drives, etc.) can be shared.

Through the use of extensive administrative support tools, each computer controls what resources it makes available for sharing and what remote resources it makes available to local users. Furthermore, security features are provided to enable a local system to accept or deny access to local resources on a per remote system and per remote user basis.

The sharing of a resource (a directory and its contents) is accomplished through the use of a location-independent mount mechanism. Once the sharing of a directory has been established, users can access its contents in the same way they would access locally resident resources. Full UNIX file system semantics, including the semantics of File and Record Locking operations, are preserved when accessing remote resources.

The RFS software has been written to be independent of the network medium and protocol and will run over any transport provider (a transport provider is any network implemented to conform to the STREAMS Transport Provider Interface, described below).

Benefits: RFS provides users with the ability to connect multiple computers via a network and share both files and peripherals. It allows users to access information in file systems of other machines on the network as if it was in a local file system, providing a simple way of sharing information across machines. Users do not have to learn commands to send files across the network and do not have the problem of trying to maintain consistency between multiple copies of the same data. Costly peripherals such as laser printers, plotters, and tape drives can be shared.

Since RFS preserves UNIX file system semantics and maintains compatibility of the system call interface relative to previous releases, binary compatibility of application programs can be preserved across operating system releases. This means that existing application programs can be installed and executed in an RFS environment. In addition, these programs can access remote files. Note also that applications that use the File and Record Locking mechanism will protect files from concurrent access by two or more (remote) systems, thus preserving data integrity.

STREAMS:

The STREAMS feature is a mechanism which provides a uniform method of implementing network protocols and supporting different network media. It enables the development of application software that is independent of the underlying network. A change in medium or protocol can be accommodated through the substitution of STREAMS modules without need for modification of the application software. It also enables the development of protocol software in

* Registered trademark of AT&T in the USA and other countries.

simplified by the ability to combine STREAMS modules to perform more sophisticated network services and to use the same modules over different media and in different network architectures. The STREAMS feature is provided in addition to the existing device driver mechanism.

Benefits: The main benefit of the STREAMS mechanism is that it provides network protocol and media independence when combined with the Transport Level Interface. AT&T and third party vendors are developing compatible STREAMS-based networks and network applications.

Transport Level Interface (TLI):

The TLI specifies user-level functions that provide access to standard protocol services defined in the ISO Transport Service Interface (i.e., reliable peer-to-peer communication across an arbitrary concatenated network). The Transport Provider Interface (TPI) specifies capabilities that must be provided by a STREAMS-based transport provider and the interface to those capabilities required in order to maintain consistency with the TLI library. Taken together, they provide the means by which network-independent applications can be written. An application written using the TLI library will work without modification over any network implemented according to the TPI specification (any transport provider). The Remote File Sharing feature is implemented using TLI.

Benefits: The specification of these two interfaces and adherence to them by developers of network services provides end users with the ability to run applications independent of the underlying network. For example, an end user running Remote File Sharing over a particular network today will be able to substitute another network, as their needs change, without losing their investment in application software.

UUCP Family:

As a part of UNIX System V, Release 3.0, the uucp family of commands (uucp - file transfer, uux - remote execution, cu - terminal emulation) have been reimplemented to use the TLI library. These changes make the uucp family independent of the underlying network and usable across all transport providers. It is no longer necessary to provide different file transfer commands for different networks.

Benefits: This command is another example, in addition to RFS, of an application written to take advantage of the network independence provided by the Transport Interface. Users can use the uucp command over any transport provider.

Shared Libraries:

Shared Libraries allow a binary application to be dynamically linked to an executable library function at runtime. Significant disk and memory savings can result from the use of shared libraries. Functions in a shared library are stored only once on a disk and once in memory, and are shared by all executable files (on disk) and all processes (in memory) that use them. In addition to space savings, the use of shared libraries reduces the effort required to incorporate corrections or enhancements made in shared library functions. When a shared library is updated, the updates are automatically applied to any binary files that access the library. UNIX System V, Release 3.0 includes a subset of libc and the full TLI library as shared libraries. In addition, users can purchase separately the tools to allow them to create their own shared libraries.

Benefits: Use of shared libraries results in a direct savings of disk space and memory.

Additional Enhancements:

Additional enhancements are also provided with UNIX System V, Release 3.0. They include increased standards conformance, improved terminal support and changes to the UNIX System signal mechanism.

AUUG

Membership Categories

Now that the Australian UNIX systems User's Group has existed a while, its time that all members reviewed their membership types, and even more, checked that they are in fact still members!

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 over 2 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 AUUGN

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.

If your membership has already expired, or is about to expire (many expire in January) then now is the time to renew.

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.

Robert Elz

AUUG Secretary.

AUUG

Application for Ordinary, or Student, Membership Australian UNIX* systems Users' Group.

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

To apply for membership of the AUUG, complete this form, and return it with payment in Australian Dollars 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 an invoice. Foreign applicants please send a bank draft drawn on an Australian bank, and remember to select either surface or air mail.

I, do hereby apply for

- Renewal (indicate which membership type).
 Membership of the AUUG \$ 50.00
 Student Membership of the AUUG \$ 30.00 (note certification on other side)
 International Surface Mail \$ 10.00
 International Air Mail \$ 50.00

Total remitted

AUD\$ _____

(cheque, money order)

I agree that this membership will be subject to the rules and by-laws of the AUUG as in force from time to time, and that this membership will run for 12 consecutive months commencing on the first day of the month following that during which this application is processed.

I understand that membership includes a subscription to the AUUG newsletter, and that I will be entitled to attend AUUG sponsored functions at member rates for the duration of my membership.

Date: ____ / ____ / ____ Signed: _____

Tick this box if you wish your name & address withheld from mailing lists made available to vendors.

For our mailing database - please type or print clearly:

Name: Phone: (bh)

Address: (ah)

..... Net Address:

.....
.....
..... Write "Unchanged" if details have not
..... altered and this is a renewal.

Student Member Certification (*to be completed by a member of the academic staff*)

I, certify that
..... (name)
is a full time student at (institution)
and is expected to graduate approximately ____/____/____.

Title:

Signature:

Office use only:

Chq: bank _____ bsb _____ - _____ a/c _____ # _____

Date: ____/____/____ \$

Who: _____

Memb# _____

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 to:

AUUG Membership Secretary
P O Box 366
Kensington NSW 2033
Australia

Foreign applicants please send a bank draft drawn on an Australian bank, and remember to select either surface or air mail.

..... does hereby apply for

<input type="checkbox"/> Renewal of existing Institutional Membership	\$250.00
<input type="checkbox"/> New Institutional Membership of the AUUG	\$250.00
<input type="checkbox"/> International Surface Mail	\$ 20.00
<input type="checkbox"/> International Air Mail	\$100.00
Total remitted	AUD\$ _____ (cheque, money order)

I agree that this membership will be subject to the rules and by-laws of the AUUG as in force from time to time, and that this membership will run for 12 consecutive months commencing on the first day of the month following that during which this application is processed.

I understand that I will receive two copies of the AUUG newsletter, and may send 2 representatives to AUUG sponsored events at member rates, though I 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 complete the other side.

Please send newsletters to the following addresses:

Name:
Address:
.....
.....
.....
.....

Name:
Address:
.....
.....
.....
.....

Write "unchanged" if this is a renewal, and details are not to be altered.

Please indicate which Unix licences you hold, and include copies of the title and signature pages of each, if these have not been sent previously.

Note: Recent licences usually revoke earlier ones, please indicate only licences which are current, and indicate any which have been revoked since your last membership form was submitted.

Note: Most binary licensees will have a System III or System V (of one variant or another) binary licence, even if the system supplied by your vendor is based upon V7 or 4BSD. There is no such thing as a BSD binary licence, and V7 binary licences were very rare, and expensive.

- | | |
|--|--|
| <input type="checkbox"/> System V.3 source | <input type="checkbox"/> System V.3 binary |
| <input type="checkbox"/> System V.2 source | <input type="checkbox"/> System V.2 binary |
| <input type="checkbox"/> System V source | <input type="checkbox"/> System V binary |
| <input type="checkbox"/> System III source | <input type="checkbox"/> System III binary |
| <input type="checkbox"/> 4.2 or 4.3 BSD source | |
| <input type="checkbox"/> 4.1 BSD source | |
| <input type="checkbox"/> V7 source | |
| <input type="checkbox"/> Other (<i>Indicate which</i>) | |

Office use only:

Chq: bank _____ bsb _____ - _____ a/c _____ # _____

Date: ____ / ____ / ____ \$

Who: _____

Memb# _____

AUUG

Application for Newsletter Subscription Australian UNIX* systems User Group.

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

Non members who wish to apply for a subscription to the Australian UNIX systems User Group Newsletter, or members who desire additional subscriptions, should complete this form and return it to:

AUUG Membership Secretary
PO Box 366
Kensington NSW 2033
Australia

Please don't send purchase orders — perhaps your purchasing department will consider this form an invoice. Foreign applicants please send a bank draft drawn on an Australian bank, and remember to select either surface or air mail.

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:

- | | |
|---|----------|
| <input type="checkbox"/> Subscription to AUUGN | \$ 50.00 |
| <input type="checkbox"/> International Surface Mail | \$ 10.00 |
| <input type="checkbox"/> International Air Mail | \$ 50.00 |

Copies requested _____

Total remitted

AUD\$ _____

(cheque, money order)

Tick this box if you wish your name & address withheld from mailing lists made available to vendors.

Office use only:

Chq: bank _____ bsb _____ - _____ a/c _____ # _____

Date: ____ / ____ / ____ \$

Who: _____

Subscr# _____

AUUG

Notification of Change of Address

Australian UNIX* systems Users' Group.

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

If you have changed your mailing address, please complete this form, and return it to:

AUUG Membership Secretary
PO Box 366
Kensington NSW 2033
Australia

Please allow at least 4 weeks for the change of address to take effect.

Old address (or attach a mailing label)

Name: Phone: (bh)

Address: (ah)

.....
.....
.....
.....
.....

Net Address:

New address (leave unaltered details blank)

Name: Phone: (bh)

Address: (ah)

.....
.....
.....
.....

Net Address:

Office use only:

Date: ____ / ____ / ____

Who: _____

Memb# _____

AUUG

Annual Elections 1987

The AUUG Committee Elections for 1987 are due soon, but before we can have an election we need some candidates!

Now is the time to nominate, or get yourself nominated, for one or more of the committee positions.

You will find a suitable form (though actually anything carrying the required information will do) in this issue of AUUGN.

Nominations require 3 financial members of AUUG to sign, plus the signature of the member being nominated.

One form can nominate a member for several committee positions, the elections are held in the order listed on the form, a candidate elected to a position is then ineligible to be elected to positions not yet decided.

Completed nomination forms should be returned to

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

to reach there no later than April 30, 1987.

Robert Elz

Honorary Secretary,
AUUG.

Australian UNIX* systems Users' Group

Annual Elections – Nomination Form

We,
..... and

.....
being current financial members of AUUG do hereby nominate

.....
for the following position(s)**

President
Secretary
Treasurer
General Committee Member (4 to be elected)
Returning Officer
Assistant Returning Officer
Auditor

Signed Date

.....
.....
.....

I,
do hereby consent to my nomination to the above position(s).

Signed Date

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

** Strike out position(s) for which nomination is **not** desired. Each person may be elected to at most one position, and the ballot for positions shall be determined in the order in which the positions are listed on this nomination form.

Nominees should use the space below (and over the page if necessary) to provide a short statement of policies, aims and/or objectives to be circulated to members of AUUG at the time of the election.