AUUGN

Australian Unix systems User Group Newsletter

Volume 9 - Number 5

October 1988

Registered by Australia Post Publication No. NBG6524

The Australian UNIX* systems User Group Newsletter

Volume 9 Number 5

October 1988

CONTENTS

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

AUUG General Information

Memberships and Subscriptions

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

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

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

General Correspondence

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

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

AUUG Executive

President	**Greg Rose** *greg@softway.sw.oz* Softway Pty. Ltd., New South Wales	Secretary	**Tim Roper** *timr@labtam.oz* Labtam Limited, Victoria
Treasurer	**Michael Tuke** *no net address* Edge Computer, Victoria		
Committee Members	**Frank Crawford** *frank@teti.qhtours.oz* Q.H. Tours, New South Wales		**Richard Burridge** *richb@sunaus.aus.oz* Sun Microsystems Austrlia New South Wales
	Chris Maltby *chris@softway.sw.oz* Softway Pty. Ltd., New South Wales		**Tim Segall** *tim@hpausla.aso.hp.oz* Hewlett Packard Australia, Victoria

Next AUUG Meeting

Regional Meetings will be held during February 1989, and the major Conference and Exhibition, AUUG89 will be held at the Sydney Hilton Hotel from Tuesday 8th to Friday 11th August 1989. Further details are provided in this issue.

AUUG Newsletter

Editorial

Welcome to the Newsletter.

I have resigned as Editor of the Newsletter and gave my resignation to the Committee at the AUUG88 Conference. Before you start rejoicing, I have given 12 months notice to allow time for a new Editor to be found and installed. This time will see me out as Editor until the August 1989 Issue.

I would like to see my last six issues as Editor be a fitting finale For this too happen, must I say it again, I need people to contribute, so

GO TO IT

Also if there are any budding Editors out there don't hestitate to put your name forward - I am sure you find the work rewarding.

Thanks to those of you that send me articles and contributed to the production of issue - it is very much appreciated.

I hope you enjoy this issue and please feel free to contribute an article soon.

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

ALSO, please find enclosed with your copy an order form for the USENIX Journal *Computing Systems*. Institutional Members already receive a copy of the Journal automatically.

AUUGN Correspondence

All correspondence reguarding the AUUGN should be addressed to:-

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

ACSnet: *john@wcc.oz*

Phone: +61 3 764 1100

Contributions

The Newsletter is published approximately every two months. The deadline for contributions for the next issue is Friday the 16th of December 1988.

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

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

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

Advertising

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

Mailing Lists

For the purchase of the AUUGN mailing list, please contact Tim Roper.

Back Issues

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

Acknowledgement

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

Disclaimer

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

President's Report

October 1988

Since the last newsletter, containing the bulk of the AUUG '88 conference proceedings, was produced, a number of things have happened within the group.

Firstly, the conference itself actually happened. The three days in Melbourne seem to have been a resounding success, with the event proceeding smoothly, and good reports from attendees, speakers and exhibitors alike. Gelato appears to be a good lubricant (and if you are not sure what that means you really must come to the next one).

The committee has made some progress toward organising the membership database better and combining that with part time secretarial service. This should ensure that the membership is serviced more promptly and conveniently for all concerned, and give us a mechanism for growing the group both faster and more smoothly.

I am pleased to say that the West Australian Unix User's Group, previously independent of the main group, has become the first chapter of AUUG. This process occurred recently when Ken Thompson was invited to Perth to speak for them. In the two weeks between confirming a date and speaking, this regional chapter managed to get about 120 people together for a buffet lunch and talk. This too is reported to have been very well received.

To really fulfil the aims of the new conference organisation, it is important that regional meetings like that in Perth should be organised in other places as well. And not just the remote places either. I believe that both New South Wales and Victoria should also have regional chapters and summer meetings. But rather than continuing to have the same faces in those groups I am calling for volounteers to organise those chapters. It's really easy guys, hardly any work at all...

We also need an understudy for the newsletter editor; John Carey will be a hard act to follow, so you should start now.

That is probably enough exhortation for one report, so until the next issue,

regards,

Greg Rose.

Secretary's Letter

AUUG88 has been over for exactly a month, life has returned to its normal level of chaos and I can wear my Secretary's hat a bit more often. Thanks again to all those who contributed to a successful conference and to those who have offered kind thoughts and constructive criticism. New lessons are learned with every conference; hopefully the same lessons are not re-learned too often.

Plans for AUUG89 are underway, with a timetable that we hope will lend itself to new features, more publicity and higher attendances. Please note the details in next year's diary as soon as you get it. AUUG89 will be held at the Sydney Hilton Hotel from Tuesday 8th to Friday 11th August, 1989, with the Tuesday devoted to tutorials.

New ideas for social events, competitions and technical demonstrations are welcome, especially when accompanied by an offer to organise. To continue to improve, the conferences need input from more people with a wider range of skills. For example, we need marketing and accounting people to play a role in the planning of AUUG89.

The next major activities of AUUG will be the local, technical meetings in early 1989. Although the Management Committee will provide funding and guest speakers, the existance of these meetings relies on the willingness of people to take care of the local arrangements. One of the reasons for having this kind of event is that the organisational effort should be relatively small. With no exhibition and with informal arrangements for meals and accommodation, if any, the local organisers will need to arrange the meeting place, audio-visual equipment and perhaps coffee and things.

Perhaps these local meetings will see the formation of more local Chapters of AUUG?

Tim Roper
AUUG Secretary

Adelaide UNIX Users Group

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

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

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

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

Dennis Jarvis,
Secretary, AdUUG.

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

PHONE: +61 8 268 0156

UUCP: {decvax,pesnta,vax135}!mulga!aegir.dmt.oz!dhj
ARPA: dhj%aegir.dmt.oz!dhj@seismo.arpa
CSNET: dhj@aegir.dmt.oz

WAUG
Western Australian UNIX systems Group
PO Box 877, WEST PERTH 6005

Western Australian Unix systems Group

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

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

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

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

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

Glenn Huxtable,
Membership Secretary, **WAUG**

AUUG Incorporated

1988 Annual General Meeting

15th September, 1988
Southern Cross Hotel

MINUTES

[*Secretaries note: these minutes are subject to amendment at the next General Meeting of the Association.*]

The meeting opened at 17:07 with the entire committee and some number of members and visitors estimated by the Secretary at 150 present. The President took the chair.

1. Apologies
There were no apologies.

2. Minutes of last meeting (August 27, 1987)
A copy of the minutes of the previous meeting, being the 1987 Annual General Meeting, was projected onto the screen. Moved John O'Brien, seconded Burn Alting **That the minutes of the previous AGM be accepted.** Carried.

3. Business arising from Minutes
Item 13.
Greg Rose pointed out that the decision by the Management Committee to unbundle the price of the dinner at conferences as announced at the last AGM had been reversed. There was some discussion and a straw vote was taken that indicated that the vast majority preferred bundling. Scott Colwell pointed out that the sample was biased. Greg Rose suggested that a majority of members appeared to be present anyway.

4. Returning Officer's Report
John O'Brien reported that for the May 1988 election and referendum:
(a) there were 180 members eligible to vote
(b) there were 72 ballots returned, one being informal
(c) the results of the election were as published in **AUUGN** volume 9 number 4, namely Greg Rose as President, Tim Roper as Secretary, Michael Tuke as Treasurer, Rich Burridge, Frank Crawford, Chris Maltby and Tim Segall as general committee members, John O'Brien as Returning Officer. No candidates remained for the position of Assistant Returning Officer.
(d) the referendum was passed with one abstention and all others in favour.

5. President's Report
The President, Greg Rose, expressed his thanks to the outgoing committee and introduced the new committee. Moved Rich Burridge seconded Geoff Cole **That the President's report be accepted.** Carried unanimously.

6. Secretary's Report
The Secretary, Tim Roper reported as follows:
(a) The group currently has 41 Institutional, 4 Student, 0

Honorary Life and 202 Ordinary members and 17 newsletter subscriptions. There was some discussion on the low numbers of Student members. John Lions suggested that students were no longer studying UNIX *per se* to the same extent as in previous years.

(b) The Secretary has been exploring various ways of improving the standard of membership processing by employing secretarial assistance and regretted that no choice has yet been made.

(c) The previous meeting of the Management Committee (12/9/88) had appointed a sub-committee chaired by Greg Rose to co-ordinate regional, technical meetings in early 1989.

(d) The 1989 Winter Conference and Exhibition will be held at the Hilton Hotel in Sydney on Tuesday 8th to Friday 11th August, 1989, with tutorials being held on the Tuesday.

(e) The Editor of **AUUGN** has indicated that he needs more articles for publication and that appointing sub-editors with responsibility for special sections may help.

(f) **AUUG88** has approximately 290 registrants, including 250 in advance, and 20 exhibitors. New features include a membership desk and published proceedings. The next issue of **AUUGN** would contain some papers that were not available at the proceedings deadline. Thanks were due to Wael Foda of Australian Convention Managements Services for organising the exhibition, sponsorships, hotel, registrations and accommodation.

Moved John Lions, seconded Glenn Huxtable **That the Secretary's report be accepted.** Carried.

7. Treasurer's Report

Michael Tuke summarised the balance sheet prepared by the retiring Treasurer and read out the letter from the accountant David Howe (both attached). He stated that the Management Committee had moved to engage the services of an accountant to keep and audit the books in the future.
Moved Geoff Cole, seconded Andrew Worsley **That the Treasurer's report be accepted.** Carried.

8. Meetings in 1989

This item had been covered in the Secretary's report.

9. Next Annual General Meeting

Moved Tim Roper, seconded Peter Tyres **That the next Annual General Meeting of the group be held at the Hilton Hotel in Sydney on the 10th August, 1989 at 17:00.** Carried unanimously.

10. Other Business

(a) Moved John Carey, seconded David Purdue **That this meeting be held despite the short notice.** Carried unanimously.

(b) Greg Rose announced that the committee had accepted the resignation of the Editor of **AUUGN**, John Carey, effective in 12 months time. Moved Greg Rose **That John Carey be thanked.** Carried by acclamation.

(c) There was general discussion of alternative ways of performing the current Editor's functions.

(d) A question was asked about the intended processing of the face images being collected at **AUUG88**. Chris Maltby answered.

(e) Peter Tyres suggested that **NZUSUGI** felt that previously good relations with **AUUG** had deteriorated. Chris Maltby suggested that this impression may have been gained because our (paper) mail to them was being returned due to their having changed address. Tim Roper pointed out that there was an official delegate from **NZUSUGI** at **AUUG88**, as there was from **UKUUG**.

Moved Chris Maltby, seconded John O'Brien **That the meeting be closed.** Carried. The meeting then closed at 17:55.

Minutes from the AUUG ACSnet SIG Meeting 15th September, 1988

The meeting opened at 0900 with Chris Campbell as the Chairman.

Ron Baxter presented the results of an ACSnet traffic survey which was conducted earlier in 1988. This survey was conducted over a period of one week.

The survey collected responses from 110 sites with data involving 399 sites and 918 links. A total 1,113 Mb of data was transferred during the survey's period, although the question was posed, what percentage of the total is this figure? In analying the data, postcodes were used as geographic handles and these were then aggregated into large regions. It was seen that traffic on inter–city links was 126 Mb and the largest of these was the Sydney–Melbourne link which carried 36 Mb. It was also seen that many links were dominated by news.

A pictorial representation of inter–city traffic was shown and appears at the end of these minutes.

Bob Kummerfeld then reported of the growth of ACSnet's "backbone" of leased lines. A year ago, most of these were dialup links but now are leased lines of one form or another. Links such as Sydney–Melbourne, Melbourne–Adelaide, Melbourne–Perth and Sydney–Wollongong are in place and proposed links such as Sydney–Brisbane and Sydney–Armidale are currently being organised. Bob proposed that these links should run the Internet Protocol (eg SL/IP) which would provide better services and connectivity. He also proposed that if IP were to be used then it would be preferred that links be 9600 baud or greater. A backbone connecting major cities would cost in the region of fifty thousand dollars ($50,000) per year.

Bob then talked about the AVCC/ACDP Report. Two years ago Queensland University proposed an Australia wide academic network based on the ColourBook protocols and DEC hardware. Last year a proposal was put to the AVCC to establish a working party to investigate this network. Earlier this year, the ACDP was included and Dr Brian Carrs from Queensland was appointed by the AVCC/ACDP committee to carry out a survey and report back to the AVCC/ACDP. The survey was send to Computing Centre directors only. Bob was invited to talk to the working party regarding ACSnet. This he did. One of the points raised during these discussions was that the working party was worried about the commercial sites within ACSnet.

The report was disappointing. It said that ACSnet was the largest network but lacked performance in message transfer timing and that there was no indication of improvement of ACSnet.

The report's conclusion was that a self-funding 2Mb backbone be put in place. This backbone was to carry both voice and fax as well as data. Each trunk on the backbone was to be split into 64 Kb channels and that X.25 run over each. The report recommended that the first year should be devoted to establishing the management for the network, allocating $270,000 for salaries, and that in the second year the backbone be put in place.

The report had some technical flaws and it was suggested that the report's recommendations took the wrong approach, in that, although "thinking big", current traffic didn't warrant such a wide bandwidth and the proposed management. It was suggested that the AUUG should, as a body, formulate and submit a reply to this report.

The following suggestions were made –

- The AUUG should not get "off–side" with the AVCC/ACDP as we are both striving for the same goals.
- The AUUG should call a halt to the current plans (management then implementation) and submit a better design – for example, include Tasmania.

- A solely academic network is crazy, why not include research and commercial organisations. We must foster a complete network!
- The network should cater to all groups and we should join together with other networks (eg SPEARNET) and produce a new report.

The meeting was then open to the floor for comment, some of which appear below

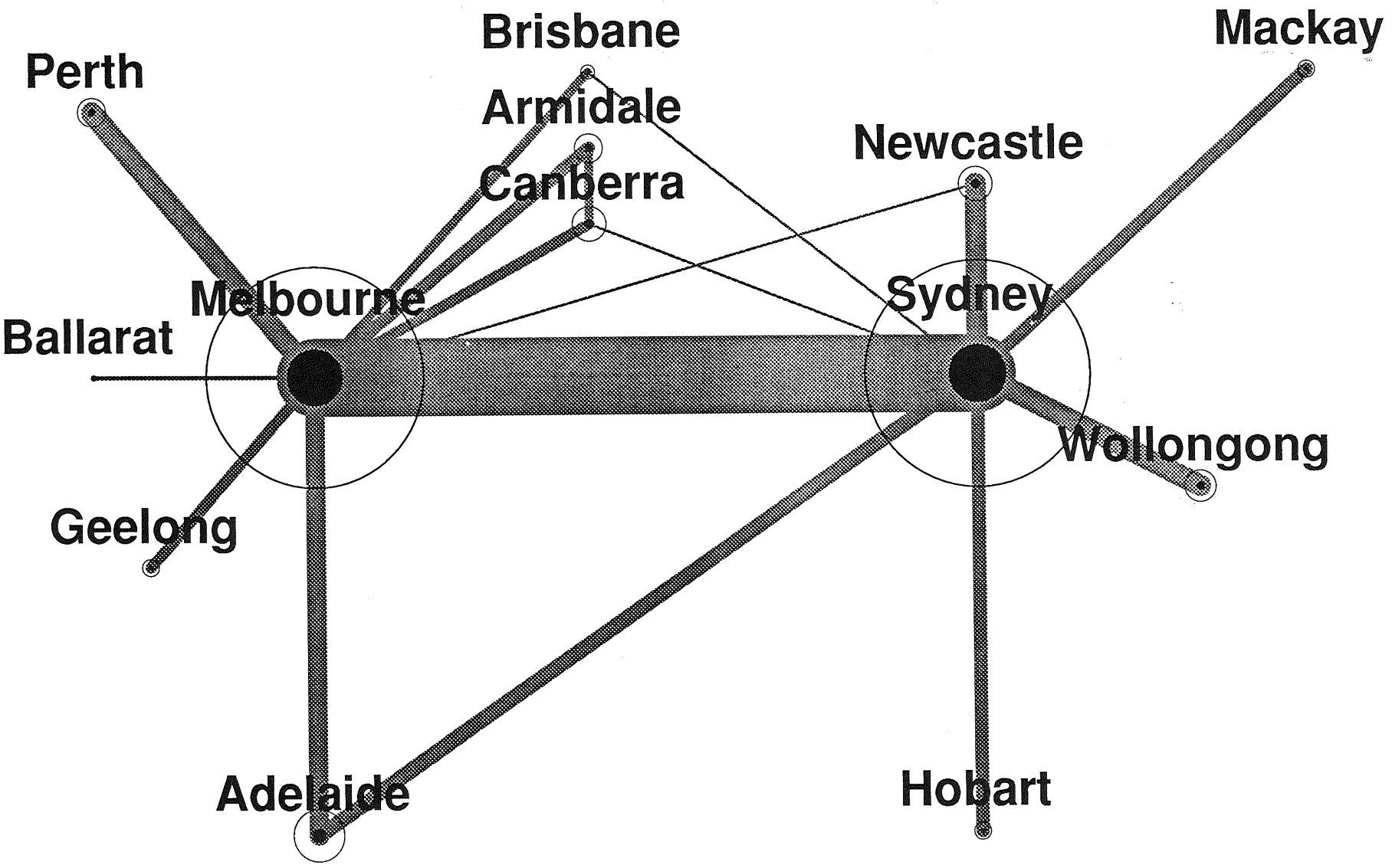
- It was reinforced that the AVCC/ACDP proposed too much bandwidth.
- A question was raised regards the design, in that, what type of machine would act as the router in each major site?
- It was pointed out that this network had to cater to more operating systems than just Unix.
- The legal implications of such a network had to be investigated.
- It was pointed out that libraries wanted to use the network and their use would be predominantly SNA. So perhaps large bandwidth is necessary.
- If voice, video and fax are to be included then the 2.0 Mbit bandwidth is needed but the network should grow slowly.
- JANET, when first established, overloaded very quickly and the time needed to upgrade was insufficient to meet needs. It was also pointed out that Unix ColourBook doesn't work!
- Bob Kummerfeld was appointed to represent the AUUG.
- We should lobby the AVCC/ACDP now as they are about to meet regarding the network proposal.
- A member from the ACDP said that their group did not support the trunk network as yet, although they do support a core management team.

A set of resolutions from this meeting were made and appear below.

- We support the initiative of having an Australian wide networking environment for the Australian community.
- A moratorium of 2 months be placed on the current AVCC/ACDP report. During this time period no money or people will be allocated for the current AVCC/ACDP proposal for an Australian network.
- A network proposal should be extended and developed to encompass the academic (Universities, TAFEs, CCAEs, etc), research (CSIRO, DSTO, etc) and commercial communities.
- A new proposal should be developed during the next 2 month period.

The meeting ended.

ACSnet survey: inter-city traffic



AUUG Conference Report

Keith B. Lewis Kathy Ching

October 13, 1988

1 The Conference

The 1988 Australian UNIX Users' Group Conference was held at the Southern Cross, Melbourne, on the 13^{th}, 14^{th} and 15^{th} of September, at a cost of $200 per person. It was attended by about 300 attendees, rather a lot of whom were wearing suits, although Greg Rose wore a tee shirt with '`/nev/dull`' written on it.

2 The Exhibits

There were about 20 companies exhibiting products at the Conference. IBM had two stands. One was exhibiting a rather boring looking minicomputer with some impressive display terminals, and the other was exhibiting an RT running AIX supporting a number of dumb terminals. This made IBM look a bit behind the times as most of the other vendors were showing work stations or PC's, but KBL was impressed with the range of commercial software running on the RT and the simplicity of setting it up.

Pyramid had a stand with a boring minicomputer and a 386 based box capable of running UNIX and DOS programs simultaneously.

The SONY stand had the most eye-catching workstation display. Their boxes were displaying a picture of a bowl and chopsticks that changed colours very impressively.

Comperex had a display of an RISC based UNIX box many times the power of a 780, not much bigger than a PC, mostly air inside, and with XT and AT expansion slots. They also had a video camera that was used to get images of most of the attendees. The purpose of this is that the images will be distributed to those who have workstations, and when such people receive mail from another person who also attended the Conference, a picture of the person who sent the mail will appear on the workstation screen. They also gave away free gelati during the breaks between sessions. They won't be forgotten in a hurry.

Other exhibitors included HP, Prime, NCR, NIXDORF, SIGMA, SUN and Honeywell. They all had UNIX boxes, workstations and PC clones.

One of the vendors actually had a VAX running VMS at their stand but they did their best to keep it hidden.

3 Formal Sessions

There were about nine formal sessions each day. They were given by such illustrious speakers as Ken Thompson (he and Denis Ritchie invented UNIX), Michael Lesk (he invented UUCP, -ms, tbl, lex and co-authored learn), Mike Karels (research leader at Berkeley) and John Mashey (VP MIPS Computer Systems).

3.1 Day 1: 13^{th} Sep

The Conference was opened by Prof Poole (Com Sci Melb Uni) who gave a short history of UNIX in Australia. (Wollongong started it, Robert Elz brought it to Melbourne and it grew.)

The AUUGN printed copies of nearly all the papers presented, so I'll just describe those briefly, but the first paper was very good and relevant, so I'll describe it in some detail. It was given by Michael Lesk and described the origins of UUCP and purported to show that informal user supported networks are better than centrally administered ones.

Michael was a software support person to whom people would bring their computing problems. He analyzed the types of problems brought to him. Most of them were solved by telling the user to read the manuals. Of the rest, most could be solved by installing the latest version of software on the machine that the person with the problem used. These things got in the way of the interesting problems. His response was to invent UUCP as part of a master plan to do automatic software upgrading on remote machines. Mostly this was thwarted by administrators who took a dim view of him mucking about with their software, and claimed it was a security problem. Actually it wasn't as big a security risk as the people with problems giving him their usercodes and passwords.

UUCP grew, and became the basis of UUnet and thus is a kind of ancestor of modern networking.

He then discussed some other networks:

ARPANET 150 sites, 56Kbit/sec links.

BITNET 1306 sites, 9.6Kbit/sec links.

UUCP >7000 sites, 1.2Kbit/sec links.

He then discussed the uses of networks, and in particular Email. A survey of Email at Bell Labs showed:

- messages average 70 words.
- 85% of Email is local.
- average message is received within 2.5 hours of being sent.
- average number of addressees is 1.2.
- 70% of mail is someone asking to be reminded of something that he once knew.

Lessons he claims to have learned:

- Don't have high speed and low speed links to the same destination. People will get used to the high speed link. If it fails it's better to wait for it to be repaired than to use the slow link.
- If dialout attempts fail, increase the time waited before trying again. Otherwise you get very annoyed phone owners when the computer is dialling the wrong number.
- Informal user supported networks are better than centrally administered ones. The network administrators may or may not own the network, but they do not own the computers attached to it. With the proliferation of systems it becomes impossible to get *all* the system administrators to agree to anything.
- Security is not worth worrying too much about. It's better to just make sure everything is identified. Then you can jump on offenders at a later time.
- low cost is worth bad performance.
- Separating E-mail from computing fails. There have been serious attempts to stamp out netnews.
- UUCP is not an entirely good thing. It is partly responsible for net-noise.
- Networks may go the same way as phonographs and television. All three were invented as educational gadgets.

- It is useless to fight the forms... We have to kill the people producing them. (This was a quote from an anti USSR propaganda slogan.)
- (He gave an example of queueing involving railways. Obviously an enthusiast)

The second paper was on future telecom products presented by Steve Jenkin.

- Telephone, Fax are here to stay.
- Telegram, Telex are evaporating.
- Viatel is going fairly well.
- X400 is coming.
- ISDN is coming, but no-one at the conference seemed to be happy about this. Only Telecom was seen as getting anything out of it.
- MAN's are coming. These are 150 Mbit/sec fibre metropolitan networks.
- BISDN — Don't hold your breath. This involves 4 150Mbit/sec fibers to every subscriber.

A company called QSPX was highly praised for work in this area.

There were two papers given on SUN-IV, the next version of the ACSnet software, by its developers.

- Some OSI compatability.
- Different routing algorithm.
- Faster and better. Especially at startup of high speed links.
- Simplified management.
- Fast or cheap delivery.
- X400 message handling.

The next three papers were on windowing. One was on NeWS, SUN's postscript based windowing system. This is based on the concept of processes. An application program opens a channel to a workstation, and then sends it commands in a superset of postscript. Some of these commands set up processes that wait for events, such as 'right mouse button while over canvas A'. Other processes are set up to redraw canvases if they become 'damaged' by occlusion etc. 9600 baud is quite ok for this sort of windowing because of the expressive power of postscript.

X-Windows is a quite different sort of windowing system. It is based on collection of 258 primitive routines, a library of less primitive routines and some higher level objects, e.g. widgets. It is available free from MIT.

Greg Rose of Softway presented a most amusing paper on the only windowing system he could afford.

- He couldn't afford a BLIT.
- He couldn't afford a SUN or an Apollo.
- He tried two terminals side by side but that failed.
- He tried a PC running GEM with multiple sessions, but that failed.
- He finally tried an Atari ST running some free software he got from the UK. It works!

Next came a paper about implementing OSI protocols, especially FTAM. Results showed nearly an order of magnitude increase in code size between FTP and FTAM.

After that Mike Karels from Berkeley presented ISO/OSI under Berkeley Unix. The slides of his talk will be printed in the next issue of AUUG.

3.2 Day 2: 14^{th} Sep

John Young of SUN, Larry Crume of AT&T, Ross Bott of Pyramid and Tom Daniels of HP discussed the SUN/AT&T based UNIX vs the Open Software Foundation alternative.

- System 5 will be merged with Xenix and SUN-OS to produce System 5 V 3.4.
- Application Binary Interface (ABI) was mentioned. This enables companies to sell 'shrink wrapped software' as is currently done with PC's.
- Application Source Interface (ASI) was also mentioned. This might enable portable programs to be written.
- Everyone seemed to agree that UNIX is now out of the 'small is beautiful' stage. Database systems are getting written into the kernel, along with whole hosts of other features. Experimentation is going on in the area of add ons rather than UNIX itself.

The System V version 4 directions are :

- In the operational, administration and maintenance areas are - backup and restore commands, configuration management, software installation, messages handling facilities and remote control.
- Improvement in real time - scheduler, general event mechanism and async. I/O.
- POSIX conformance.
- X/OPEN capabilities. - OPEN LOOK.
- Internationalization - foreign language, e.g. Japanese, error messages etc.

Users can obtain the specification of OPEN LOOK by sending request to AT&T Japanese office.
Mike Karels then talked about research going on at Berkeley:
New features in 4.3:

- Disks now have labels.
- Flexible file system.
- Kernel memory allocator.
- Better TCP/IP that has slow start and congestion control.
- Disk tables that always should have been on disk now are.

Current projects:

- routed, EGP, gated, internet nameserver and extensions, IOS/OSI network protocol support, POSIX compliant interface.
- Wait for system call.
- Generic file system with VNODES.
- Rearrange filesystem so /usr, /bin, /etc can be shared.
- CMU MOK integration.
- Memory resident file system for /tmp.
- Reasonable hardware independent for a range of configurations.
- User interface finalized.

Virtual memory design that includes :

- Define process as regions :
  - Share memory with semaphores.
  - File mapping for private and share.
  - Copy on write for fork and file mapping.
  - Light weight processes.
- Larger share address space with multi-level, paged data structure and swapped image is not preallocated.

A paper was presented on the problems with networking SUN workstations at Sydney Uni.

- People with SUN's on their desk turn them off when they go home, and hence miss getting software upgrades etc. Various automatic updating programs were tried and rejected. Current solution is to do the upgrade, and all the machines that miss it, miss out. The upgrade will be done later for the user if he ever complains of any related software problems.
- security:

1. What is on a screen can be read by anyone anywhere on the network. This is because it is also in memory (video ram) and as well as that it appears as a device, /dev/fb, which is public.
2. Owners can reboot their workstations and hence run in single user mode.
3. The SUN diagnostic monitor allows the user to modify kernel data and code.
4. There is a bug in 'Yellow Pages' that lets unprivileged users supply their own password file for a whole network.

These problems were overcome by telling the users about them, suggesting they encrypt important files and pointing out that if they damage the network they mainly hurt themselves.

Next came a good technical paper about getting NFS to go on a VAX under 4.3 BSD. It seemed a lot of work, but really a fun kernel hacking.

Following that was a paper called 'Legal and Social Implications'. The paper was much as the title suggests. KBL enjoyed the serious suggestion that Fax and Email are illegal. Someone commented that there is a case about a company being sued for $4.2 millions related to computer services.

Next Michael Lesk gave two papers about human interface issues. The first concerned a library where the users were presented with a choice of two terminals. They could either look up books using a command driven interface (much like sesame) or a completely menu driven interface (much like the HP system at Box Hill library). The command driven interface was used by 70% of first time users and a rising percentage of more experienced users. The conclusion was that the more the users know about what the commands are likely to be, the less valuable a menu system is.

The next paper concerned the extreme difficulty he had in getting management to actually sell a product he had produced, to a customer. The product was an AI system that provides instructions about getting from place to place efficiently subject to certain conditions. E.g. when driving one has to obey street laws governing one way streets, but when walking that would not apply. He showed examples of various routes through cities based on certain optimizations and compared them with the routes people would actually choose. The standard Dijkstra breadth first graph search algorithm produced very suboptimal results. The system was originally designed as an electronic yellow pages phone book. The customer who wanted to buy it was a car rental agency.

At the dinner that night John Mashey gave an amusing talk that compared a large software project to an army invading territory. The scouts were compared to software prototypes. The users were compared to natives, and they might accept the system, or they might try and slash the tyres and put gravel in the fuel tanks. Worse even than that, they might have left the area entirely before the project was finished. We thought the talk very good value as well as good entertainment, but it was slightly marred by the distraction produced by the hotel staff clattering dishes etc.

3.3 Day 3: 15^{th} Sep

The final day started with the ACSnet meeting. The AVCC, Queensland Uni and Brian Carss came in for alternating bouts of criticism and defence. Results of some measurements of ACSnet traffic were presented. Bob Kummerfield was authorized to act on behalf of the ACSnet community in negotiations with the AVCC etc.

That was followed by the second of the two ACSnet papers. The one focused on the link management - commands file, config file and call script language.

Following that came the eagerly awaited paper by Ken Thompson. He is writing a very fast C compiler. Speed is achieved at the expense of code quality by combining two of the early passes into one, simplifying the intermediate structure, and reducing the optimization to two simple passes. He doesn't mind the lesser code quality, because he spends much more time compiling than running anyway. Surprisingly, the generated code didn't seem any worse than that of the standard compiler.

Next was a paper on distributed processing in the Queensland Government. The authors have produced a "cookbook" about how to buy, install and operate a UNIX box. Basically they have a terminal network, and wish they had a computer to computer network. Many of their machines are connected but cannot even send mail to each other.

Next was a paper on time synchronization in a network. (Pyramids over Ethernet in this case). We may well run his code on our Pyramids.

Next came a paper from ICL about a small office automation network. The MICROLAN 2 cable can be used to connect up to 8 workstations and among them one is the master unit.

Next came a very good paper about how to write portable UNIX programs. This was a very valuable talk. The paper itself was published in the AUUG newsletter. There were some notable comments on writing portable socket or share memory or semaphores code. Its is not easy.

Next came a paper by a Korean visitor about some real time extensions his group are making to UNIX. He spoke well but had trouble understanding the questions put to him afterwards. He gave the impression that the Korea Government and Companies (e.g. Samsung) have spent millions in research and development in his introduction of the paper.

Next came a paper on STIX. That is a port of MINIX to the Atari ST. As a MINIX user KLB was quite interested. The system runs well and is useful but the overhead needed to move programs around in memory was a bit disturbing. It was necessary because the ST has no memory management hardware.

Finally John Mashey gave a talk about RISC chips and the software that goes with them. In particular he spoke about a MIPS chip which has pipelines that can partially stall, and a compiler that generally manages to rearrange the code so that the stall times can be utilized. e.g. The instruction after a test and branch instruction is executed irrespective of the result of the tested condition. Only after that instruction completes will the branch happen or not as appropriate.

4 Informal discussions

We had quite a lot of informal discussions with other attendees at the meeting, and this resulted in several people suggesting solutions to some of our problems, and some offering to let us use bits of their software. This will be of great value to the university.

5 The End

And that was the end of a really worthwhile and informative, but extremely intense and tiring conference. (Following it there was an unannounced 3 hour delay affecting trains on the city loop, but KBL did get home, eventually).

Keith Lewis

Kathy Ching.

Hows a Windowing System

Greg Rose

Softway Pty Ltd

ABSTRACT

This paper gives some very subjective view about windowing systems and their application to getting work done under UNIX.

Despite being based on work done quite a long time ago, most of the world seems to believe that the use of windowing to enable multitasking of the user interface and graphics to improve the ergonomics of such an interface started with the Macintosh. Even Apple, who should know better, seem to think this.

I make no apologies for my own attitude in this matter, which is that multiple task streams and environments match what *I* want to do and the way that *I* work. But I don't have a lot of time for dragging Icons around screens, or moving mice more than three metres just to get to a menu. So I make a biased and possibly incorrect assumption. Which is: that other programmers like to do things much the same as I do. Therefore this talk concentrates on an environment of programming under

What is it about the use of windows that addresses my requirements, then? It is simply the existence *within the interface to the computer* of multiple streams of commands, text, environments, or whatever, which parallels my own methods of doing work. The less obtrusive the "context switch" the better. And there are other things that I am not prepared to sacrifice even for this benefit, an example being output data rate.

One of the first examples of the sort of thing I am referring to, was the shell escape from the editor. This enabled an editing session to be interrupted by a compile, or sending mail, or whatever, without loss of the editing environment (current line, last pattern, perhaps the buffer was in the middle of some series of changes and writing the file was inappropriate).

Later came Berkeley Job Control. This enabled the act of changing environments to be relatively independent of what was actually running (i.e. the application didn't need to change to allow the equivalent of a shell escape). Having multiple virtual terminals was a different way of viewing the same functionality. But at any instant, only one terminal is really "there".

This is where windows come in. Humans are perfectly capable of assimilating input from multiple sources at once, at least to the extent of processing interrupts. For example, while reading a book to a background of music, I am perfectly capable of replying "milk and no sugar" when asked if I want a cup of tea. In the virtual terminal environment, the tea-making program doesn't talk very often, so it is unlikely to be communicating to me at any given instant.

However, it is not possible for me to carry out two conversations at once. My output stream (fingers or mouth) is definitely *not* multiple threaded.

So what I really need for my computer interface is one where I can maintain multiple environments at one time, and where any of those environments can communicate to me, even though I will only reply to one at a time.

Multiple windows, where a window may be completely visible or completely obscured, or some intermediate state, and where I can quickly change the destination of my own input, is a very good model of these requirements.

The next step is the ability to communicate b ween these environments, but that is another story.

But now that they have become popular we are faced with a number of alternatives in the human interface area, the vast bulk of which are very usable and very expensive.

I forgot to mention one other criterion for judgement. I want all of this functionality to be as cost effective as possible.

So where do most UNIX programmers stand today? I have concluded that a depressingly large number of them (you) are so used to a glass version of a hard copy terminal that the limitations of the output device are dictating artificial limitations to work practices. Most programmers think that Berkeley job control is for "controlling jobs" not for maintaining multiple environments. Very few people seem to use System V shell layers. Most people are still dreaming about having a single user workstation one day. But they are expensive, so many employers aren't providing them yet.

But those wonderful X windows or NeWS systems are designed to be able to do just about anything, like displaying a zillion different fonts or rotating a wire frame drawing in real time. These facilities are not used very often by people like me. I can do most of my job by reading and typing ASCII in just one adequate font. Why, then, do I have to pay a lot more for a terminal with all these capabilities that I don't need or want to pay for? Couldn't it be a lot cheaper?

Yes it could. Firstly, the latest whiz-bang RISC microprocessor doesn't need to be in there. For actually doing work, I prefer a centralised and efficiently shared multi user computer. After all, most of my work involves interacting with others and using shared resources. Why throw this out and then attempt to put things together to simulate it again? And if it isn't doing all of the work of a general purpose computer, then a three year old chip should be about enough to shuffle characters around a screen.

So I prefer an approach which disconnects the computing tasks from the display tasks, especially where someone tries to convince me that one-to-one correspondence is what is needed.

So how big does the screen need to be? What resolution? Well, obviously, the bigger and clearer, the better. But these are trade off items, and at the moment it is a binary trade ("Mummy, can I have a 630?" "NO!") While I would like to have a screen that can fit whole A4 pages, or fit a VT100 emulator on a postage stamp and still be readable, there is a limit to just how much that is worth. And anyway, nobody denies that 747's are good value for money, but I don't see many people with them parked in the front street. Reality has budgets.

So what I want boils down to something that can display a reasonable amount of information (at least 24x80 characters) readable for an extended period. But I also want windows, and maybe the possibility to trade readability for more size or less crowding on the screen.

The obvious solution is to have a mass produced small computer (which provides the cheapness part of the equation) and a bit of specialised software that makes it do windows sensibly. The windows should then talk mostly to another computer do do any real work.

Soft y has identified two solutions to this problem. Our first cut was to use the cheapest and nastiest PC clones we could find, and run GEM to handle windows, and a manufacturers' proprietary offerring to let those windows talk to the real computer.

The second, and much better, solution us - Atari computers, which actually have w w manipulation in ROM, and a software package from the University of Kent at Canterbury England, written by Peter Collinson.

The former gave acceptable characteristics for about $2000 per terminal. The latter gives outstanding characteristics for about $1300 (tax paid) per terminal, little more than an average glass tty.

The presentation of this paper will show some of the range of systen.. available.

I also intend to rant and rave about pop-up versus pull-down menus a bit.

The fundamentals of NeWS

Tim Long

NeWS is a system developed by SUN Microsystems for driving bitmapped graphic displays. It uses an extended PostScript to implement its drawing primitives and is designed to work in a multi-process and multi-machine environment.

NeWS attempts to combine the machine and operating system independence of the canonical VDU with the interactive capabilities of the bitmapped graphics workstation.

In short, NeWS is a system a programmer uses to implement an interactive graphic interface, in the same sense that UNIX is a system a programmer uses to implement an application.

There are two other well known systems which attempt to address some of the same issues as NeWS. One is X-Windows, the other the Blit.

X-Windows attempts to achieve device independence by defining a canonical form for classic bitmapped display operations. It also defines an encapsulation and network transfer mechanism for these operations. By this means, X-Windows attempts to drive remote and varied displays with the same sort of library routines that have been used in systems where the display is local and tightly coupled to the driving application.

Another system which addresses the 'bitmapped graphics terminal' issue is the Blit.

In addition to other innovations the Blit takes a different approach to the application-to-graphics interface by allowing an application to program the terminal. This approach greatly relieves network bandwidth and allows a well written application to give a high fidelity of interactive response. But both the Blit and X-Windows fail in important respects.

The most important is device independence.

X-Windows attempts to achieve device independence through the adoption of a canonical bitmapped display to which all operations are mapped. In reality this does not allow for much variation. If the pixels are not just about 80 to the inch things begin to look very strange. If there is any unusual special purpose hardware in the display it often has to be ignored. And the handling of colour versus monochrome is awkward. X-Windows also suffers from requiring a substantial library on the application programming machine, and has major problems with network bandwidth.

The Blit takes a more extreme attitude. A Blit is a Blit is a Blit. And if you don't have a cross-compiler and the appropriate libraries, forget it.

The Blit, X-Windows and almost all other bitmapped graphics system suffer another major problem. They are difficult to program.

The primitives provided normally start at the bit manipulation level and, despite layers of protective libraries, still require a large investment in reading, programming and debugging to make an application operate in a basic fashion. When it comes to making it slick and attractive even more work is involved. As if this wasn't enough, all systems, except the Blit, impose a structure on the application program based around a main loop with some form of 'get next event' call and an enormous switch statement to decide what to do about it. This structure is often totally inapplicable to the task at hand.

NeWS, despite years of effort by SUN, and a somewhat over-engineered implementation, solves all of these problems and more. It has the programmability of the Blit, the network transparency of X-Windows, and a device independence and ease of programming all its own.

NeWS is based on PostScript. PostScript is a stack based interpretive programming language rich in primitives for rendering static two dimensional pictures.* It is device independent and well designed for the description of printed images. The extensions which turn PostScript into NeWS revolve around:

1. The introduction of multiple processes with private stacks but otherwise shared resources.
2. The introduction of an asynchronous message passing scheme for event handling and IPC.
3. The introduction of multiple drawing surfaces and the definition of a mechanism to keep the pictures drawn on them up to date.

In practice, a NeWS terminal (or server) implements an extended PostScript (which I will hereafter call NeWS-PostScript) and operates along the following lines:

The server initially starts with one process running. This process waits for a foreign connection request (via TCP/IP say). For each of these that it receives, it forks a new NeWS-PostScript process which establishes an environment similar to the standard PostScript starting environment and begins to execute operators from the connection. Meanwhile the main process is back waiting for new connections.

The application which requested the connection is now free to send NeWS-PostScript operations down the connection. Almost all applications would first send some bunch of function definitions and initialisation code. Once in operation, the terminal (server) side of the application will have several NeWS-PostScript processes running to implement various aspects of the user-interface. Any of these may also be involved in sending data back to the main application, while one process will still be executing operators from the connection to allow the main application to control the terminal side.

---

* For a short description of PostScript see the introduction to *The PostScript Language Reference Manual* by Adobe Systems.

A slightly more detailed examination of the extensions to PostScript which turn it into NeWS will make this mode of operation a little clearer.

Memory and processes

The memory model of pure PostScript divides all objects into two classes, simple and compound objects. Simple objects are pushed directly onto stacks by value and, naturally, the memory they occupy is relased when they are popped off. But compound objects (such as arrays) are pushed onto stacks by reference and have their value in 'virtual memory'. The memory used by these objects is only reclaimed by restoring the state of the interpreter back to some previously saved state.

While in most respects NeWS is identical to PostScript, the NeWS memory model is different. Compound objects in NeWS are retained while they are referenced and freed when they are no longer referenced. But the distinction between simple and compound objects is still important when considering multiple processes.

A new NeWS process is started by a *fork* primitive. The new process gets a copy of its parent's stacks, therefore simple objects are copied, but compound objects have a value shared between the two processes because they reside in the common 'virtual memory'. Likewise drawing surfaces (the screen included) and other terminal wide resources are shared.

NeWS processes are intended to be light-weight primitives. They are used for seemingly trivial tasks. Poping up a menu or just dragging a box around the screen would typically be done by forking a process to do it. This is one of the key elements which can make user interface programming in NeWS particularly easy.

Context switching is not pre-emptive. It happens only when a process performs blocking operations or explicitly allows context switching.

Events

Events are bundles of information about something which has happened. Some are generated by the NeWS system automatically to describe real world events (such as key transitions and mouse movements), and they are also used for inter-process communication.

An event, once generated, is 'distributed' to applicable processes. Which events are distributed to which processes depends on 'expressions of interest' processes have made for events matching some pattern. For example, suppose a process wished to act on left mouse button events over a particular drawing surface. It would construct an event which looked like such an event, setting to *null* those

fields which it did not care about and specifying those fields it did (such as the drawing surface and type of event). This would then be used as an argument to the *expressinterest* operator.

A process will typically receive events by executing the *awaitevent* operator which blocks until an event of interest is available, but once one is available, it returns it.

Canvases

Canvases are things you draw on. Canvases have been introduced to allow multiple drawing surfaces as opposed to the single page of pure PostScript.

Canvases are created in a hierarchy, that is, each canvas has a parent canvas on which it was made. A canvas also has a position relative to its parent and a clipping-path (further restricted by its parent's). Unlike most other windowing systems, NeWS's canvases are bounded by arbitrary paths.

Most other properties of canvases are variable. For instance a canvas may be transparent in which case all drawing primitives are subject to its position and boundary but then just flow through to the parent. Of more utility is the opaque canvas. Painting primitives on opaque canvases directly effect the bitmap that corresponds to them (subject to obscurement by other opaque canvases).

Canvases are intended to be 'cheap' graphics primitives and should not be confused with a window.

Picture maintenance

At the heart of any windowing system is some technique for picture maintenance in the presence of opaque drawing surfaces appearing, disappearing and moving. This normally, when all else fails, involves the application being asked by the system to regenerate some image which has been lost. NeWS is no exception to this.

When some previously hidden part of an opaque canvas becomes visible (and the NeWS server does not have the bitmap hidden away somewhere) a 'damaged' event is generated for that canvas. In a correctly constructed NeWS program every on-screen opaque canvas will have some process that is interested in damaged events on it and be prepared to repaint the image. There are a number of ways of optimising this but the principle remains.

Writing a NeWS program

Writing and running programs to use NeWS requires very little support on the client machine beyond the connection mechanism to the NeWS terminal. There is a small library, the source for which is available form SUN, which implements the few routines needed to establish a connection to a NeWS server (in terms of 4.2BSD networking primitives of course). The only important function is *ps_open_PostScript()* which sets two global variables to be standard I/O streams, one to, and one from, the NeWS server.

Once a connection is established almost all NeWS client programs will wish to transmit a bunch of PostScript functions to the terminal. The obvious way of doing this is to copy the source of your PostScript functions from some data file to the connection. This is very easy to do. But for some reason which remains a mystery to me, SUN have provided a much more difficult method. SUN's documentation uses this method exclusively so you could be forgiven for thinking this is the best approach.

In brief, SUN's method involves embedding fragments of your PostScript within pseudo-C function declarations. The syntax for this is poorly designed. The resulting file is parsed by a programs called *cps* which produces an include file with declarations of arrays which contain your PostScript and macros which transmit the various fragments.

Typically one of the fragments will be the bulk of your PostScript functions and initialisations. This will normally be invoked as soon as your connection is established.

After initialisation the difference in the two methods is small. For instance, using cps to draw a line would involve defining a function in the cps file which associates a PostScript fragment with a C function, called say ps_line(), and then invoking the function. So in the cps file we would have:

```
cdef ps_line(int tox, int toy)
    tox toy lineto
```

Then in the C program:

```
ps_line(tox, toy);
```

I think it is clearer to say:

```
fprintf(PostScript, "%d %d lineto\n", tox, toy);
```

because this is all the ps_line macro does anyway.

An example

The best way to get a feel for NeWS is to try and program it, but in the absence of that, looking at an example will do.

This example is somewhat artificial because I have fully expanded (or dodged altogether) a lot of code which would normally be provided by SUN libraries. I have also not used the SUN window system. This is deliberate.

SUN's libraries seem to be designed to make programmers used to the old way of doing things feel comfortable. But in doing this they largely defeat the benefits NeWS. It may be that SUN felt it was essential to do things 'the old way' in order to gain some form of market acceptance. Their plans for the next release of NeWS which contains an X-Windows immplementation certainly suggests they feel this is important. But in my opinion using SUN's windowing system would only obscure the purpose of this example, which is to demonstrate the fundamentals of NeWS.

This example maintains the time in ctime(3) format in a small white rectangle in the lower left of the screen. The point of this simple example is that the C program (the client) knowns how to find out the date, and the PostScript side (the terminal or server) known how it wants it displayed.

The client side of the example:

1. establishes a connection to the NeWS terminal;
2. copies its NeWS-PostScript code from a text file to the terminal;
3. responds to requests for the time from the terminal.

The terminal code:

1. makes the canvas on which it will print the time;
2. forks a process which maintains the time canvas and makes periodic time requests to the client;
3. does what the client tells it to do.

For more details see the programs.

The separation of the semantics of the application from the graphics is an essential part of good NeWS programming. This simple example is based on the hypothesis that the full breakdown of the current time is something only the client application is capable of calculating, so it has the job of supplying this. But how this is displayed is entirely up to the server code.

Given the well defined format of a ctime string the PostScript side could be pulling out the numbers and using them to draw hands on a picture of Big-Ben. Whatever is was doing would make no difference to the client code.

```
#include <stdio.h>

extern FILE *PostScript;
extern FILE *PostScriptInput;
extern char *ctime();

main()
{
    FILE    *stream;
    long    t;
    char    *s;
    int             c;

    /*
     * Establish a connection to the NeWS terminal.  (Sans error
     * messages for brevity.)  This sets PostScript and PostScriptInput
     * as a side effect.
     */
    if (ps_open_PostScript() == 0)
      return 1;

    /*
     * Send the initialisation code to the NeWS terminal.
     */
    if ((stream = fopen("clock.ps", "r")) == NULL)
      return 1;
    while ((c = getc(stream)) != EOF)
      putc(c, PostScript);
    fclose(stream);

    /*
     * Do what the server side asks of us.  This is a particularly simple
     * communications protocol from the server side, but in reallity
     * you hardly ever need anything more than a scanf.  It's not as if
     * there is a user out there sending this stuff.  You contol both
     * ends so make it easy on yourself.
     */
    while ((c = getc(PostScriptInput)) != EOF)
    {
      switch (c)
      {
      case 't':
          time(&t);
          s = ctime(&t);
          s[24] = '\0';
          fprintf(PostScript, "(%s) Update\n", s);
          fflush(PostScript);
          break;

      case 'q':
          flcose(PostScript);
          return 0;
```

```
%
% I have left it to here to mention a couple of points which will
% presumable only be of interest to people who wish to read the PostScript.
%
% Process IDs, canvases, and events are all immplemented in NeWS as
% dictionaries in the same way that fonts are in pure PostScript.
% That is: a canvas is manipulated as a dictionary with well known
% element names.  Likewise for events and processes.
%
% There is an example of this immediately below where a newly
% created canvas is 'begun' in order to set some of its attributes.
%

%
% Create a new canvas (call it 'ClockCanvas') whoes parent is the framebuffer.
% Reshape it to be at 0,0 and 145 long by 16 high (in points as usual).
% Make it opaque and mapped onto the screen, then finally make it the
% 'current canvas' (and therefore the implicit target of drawing operations).
%
/ClockCanvas framebuffer newcanvas def
0 0 145 16 rectpath ClockCanvas reshapecanvas
ClockCanvas begin
    /Transparent false def
    /Retained false def
    /Mapped true def
end
ClockCanvas setcanvas

%
% This function is run as a seperate process.  It handles a few things:
% it keeps the canvas up to date, it catches clock ticks, forwards
% them to the client side and re-primes the clock, and finally it
% forwards a left mouse button press on the canvas as a finish message
% to the client.
%
/ClockProc
{
    %
    % Express intrest in damage event on the clock canvas.
    %
    createevent begin
      /Name /Damage def
      /Canvas ClockCanvas def
      currentdict
    end
    expressinterest

    %
    % Express intrest in left mouse button down transitions over the canvas.
    %
    createevent begin
      /Name /LeftMouseButton def
      /Action /DownTransition def
      /Canvas ClockCanvas def
      currentdict
    end
    expressinterest
```

```
%
% Express intrest in 'Tick' events.  These are not NeWS systems events
% but something of out own fabrication.
%
createevent begin
  /Name /Tick def
  /Process currentprocess def
  currentdict
end
expressinterest

PrimeTick                       % Start the clock.
{
  awaitevent              % Wait for anything of interest.
  %
  % The event (which we just got or we wouldn't be here) is
  % a dictionary (as usual) so 'begin' it to get easy access
  % to the contents...
  %
  begin
      /Name /Damaged eq
      {
        %
        % A damaged event, must redraw canvas.  A small amount of
        % effort (and a reasonable sense of ascetics) can make a
        % big difference here.  By changing the shape of the
        % canvas, pulling out a few numbers from the ctime string
        % and drawing hands and things you get an analog clock.
        %
        damagepath clipcanvas   % Restrict drawing to damaged areas.
        1 fillcanvas            % Erase to white.
        0 setgray
        1 4 moveto Ctime show   % Redraw the current Ctime.
                          % (The default font is 12 point Times.)
      }
      if
      %
      % For 'Tick' events we just print a 't' to the client
      % and reprime the clock.  The client will give us a
      % new Ctime value and cause us to redraw in due course.
      %
      /Name /Tick eq {(t) print PrimeTick} if
      %
      % For left mouse down event we send a 'q' to the
      % client and break from the loop (which then falls off
      % the end of this processes code causeing it to exit).
      % The other process (the main one the client is
      % talking to) will exit when the client closes the
      % connection.  This will cause our userdict to be
      % discarded which will cause the last reference to the
      % ClockCanvas to disappear which will cause it to
      % vanish from the screen and any canvases it was
      % obscuring to get damaged events.  Simple really.
      %
      /Name /LeftMouseButton eq {(q) print end exit} if
  end
}
loop
}
def

%
% Generate a 'Tick' event to arrive at ClockProc in one second.
%
/PrimeTick
{
    createevent begin
      /Name /Tick def               % "Tick" is our invention.
      /Process ClockPID def         % Send straight to the given process.
      /TimeStamp currenttime 1.166 def % TimeStamp in future, so deliver then.
      currentdict             % Leave on stack for sendevent below.
    end
    sendevent
}
def

%
% (ctime-string) Update -
%
% Update Ctime and damage the ClockCanvas (which is assumed to be
% the current canvas) to cause it to be repainted.
%
/Update
{
    /Ctime exch def
    clipcanvaspath extenddamage
}
def

/Ctime () def

%
% Fork of the ClockProc function (above) and store the pid in ClockPID.
%
/ClockPID (ClockProc) fork def

%
% This is the end of the initialisation code sent down the connection.
% But this process will continue to read anything the client program
% sends down the connection until it gets closed.  In this case the client
% will just send things like:
%
%     (Tue Aug 23 01:33:08 EST 1988) Update
%
% See the Update function above and clock.c.
```

Conclusion

This description of NeWS is incomplete. There are some aspects of NeWS which I have failed to explain. Some because I hope they are replaced (such as SUN's windowing system). Others because they are simple and work (such as colour support). But mostly because I have tried to stick to the fundamentals of NeWS, for this is where its strength lies.

The programmability of the graphics terminal with a language appropriate to the task allows for great reductions in the bandwidth required between the display and application. At the same time it gives the NeWS programmer the opportunity to improve the responsiveness or sophistication of the interface using local CPU power.

The use of PostScript as a drawing primitive is a great advantages of NeWS. Although it is not often recognised, the malleability and availability of artwork is essential to the production of good graphic interfaces. PostScript is a superbly portable picture description language. Bitmapped based systems can never achieve the same degree of flexibility.

The use of light-weight processes is also an enormous advantage in interactive graphics. In many extant graphics programs the need for the application to be on-tap to perform display housekeeping at a moments notice totally perverts their structure. Being able to farm out conceptually different parts of picture maintenance to appropriately constructed processes greatly simplifies the programming task and improve the program's behaviour. But there is one point which overshadows all of the above.

The importance of device independence can not be over emphasised. Good software has a long natural lifetime, whereas hardware tends to come and go on an annual basis. Software tied to hardware will die. It may struggle for a while, supporting the hardware that dooms it, but die it will. It is in this matter that NeWS has it all over its peers. NeWS does not operate at the bitmap level, it is far more independent of changes in display technology. It is also far more capable of taking advantage of new features. It will take some radical redevelopment of X-Windows and all X-Windows applications to handle 300dpi screens. But 300dpi screens and the CPU power to drive them are almost here. Given the necessary CPU power NeWS will handle these without difficulty. Application programs will run unmodified, and take full advantage of the resolution.

Programming on UNIX System X release Y

Stephen Frede

Softway Pty Ltd

1. intro

This paper is meant to provide a useful reference guide to anyone writing or porting programs for versions BSD version 4 release 2 (BSD4.2), BSD version 4 release 3 (BSD4.3), AT&T System V (SysV), AT&T System V release 2 (SysV.2) and AT&T System V release 3 (SysV.3). When porting programs, the usual problem is to port BSD programs to SysV, rather than vice-versa.

Note that this is a guide only, and readers should refer to the appropriate manual entries for relevant details. Also note that is is very incomplete and correspondence will certainly be entered into. Sections marked TODO will be implemented in forthcoming releases of this paper.

2. Common changes

To many people, the most common porting problem they have to deal with is to get programs from Usenet to run on their systems. In many cases, only a few changes need to be made. The most common of these are listed here for convenience. Refer to the individual descriptions given later in the paper for more details.

SysV	**BSD**
strchr( )	index( )
strrchr( )	rindex( )
#include <string.h>	#include <strings.h>
memcpy( )	bcopy( )
srand48( )	srandom( )
lrand48( )	random( )
uname( )	gethostname( )

3. System calls and library routines

All of the system calls and some of the non-compatible library routines are listed here in alphabetical order, so that they can be found easily. Descriptive parameter names are provided for some but not all of the routines as an aid to comparison more than anything else. Anything given as "pathname" is a string which refers to a relative or absolute pathname in the filesystem. The argument fd is an integer file descriptor.

accept( ) (BSD) See *Sockets*

access(pathname, mode) (any)

BSD provides definitions in <sys/file.h> not provided in SysV.

```
#if    BSD
#    include <sys/file.h>
#endif /* BSD */

#ifndef R_OK
#    define    R_OK    4
#    define    W_OK    2
#    define    X_OK    1
#    define    F_OK    0
#endif /* R_OK */
```

acct(pathname) (any)
Use is compatible. However, there are minor differences in the structure stored in the accounting file. SysV has two extra fields in the `acct` structure: `comp_t ac_rw` (no. block read/writes) and `char ac_stat` (exit status). Also the `comp_t ac_io` means no. disk I/O blocks under BSD and chars transferred by read/write under SysV. Finally, the `ac_comm` field is 10 bytes long in BSD, and 8 bytes in SysV.

adjtime(delta, olddelta) (BSD) See *Time*
This system call is used to adjust the system clock by temporarily speeding it up or slowing it down, so that the clock always gives increasing values. No comparable functionality in SysV.

alarm(secs) (any)
This SysV system call is implemented as a library routine under BSD.

bind( ) (BSD) See *Sockets*
brk(address) (any)
chdir(pathname) (any) *Compatible all versions*
chmod(pathname, mode) (any) *Compatible all versions*
chown(pathname, owner, group)
Use is compatible. Under BSD, only the super-user may change the owner of a file, while under SysV, anyone may "give away" files they already own.

chroot(pathname) (any) *Compatible all versions*
close(fd) (any) *Compatible all versions*
closedir(dirp) (BSD, SysV.3, PD) See *Directory*
connect( ) (BSD) See *Sockets*
creat(pathname, mode) (any) *Compatible all versions*
dup(fd) (any) *Compatible all versions*
dup2(oldfd, newfd) (BSD)
The BSD system call:

```
newfd = dup2(oldfd, wantedfd);
```

is equivalent to the following under either BSD or SysV

```
#include      <fcntl.h>
...
close(wantedfd);
newfd = fcntl(oldfd, F_DUPFD, wantedfd);
```

environ (any) *Compatible all versions*
execl( ) (any) See *exec*
execle( ) (any) See *exec*
execlp( ) (any) See *exec*
exect(name, argv, envp) (BSD)
Used when tracing (with *ptrace(2)*) the executed process.

execv( ) (any) See *exec*
execve( ) (any) See *exec*
execvp( ) (any) See *exec*
exit(status) (any) *Compatible all versions*
_exit(status) (any) *Compatible all versions*
f...(fd, ...)
A number of system calls exist starting with the letter "f", which correspond in function to a system call with the same name without the leading "f". The "f" system calls take a file descriptor as their first parameter. The corresponding system calls take a pathname as first parameter. Of these, fstat( ) is available on all versions, but fchmod( ), fchown( ) and ftruncate( ) are BSD specific. To emulate these under SysV, you will need to keep the pathname of the file available and use the corresponding "non-f" system call.

fchmod(fd, mode) (BSD) See *chmod( )*
fchown(fd, owner, group) (BSD) See *chown( )*
fcntl(fd, cmd, arg) (any)
Use is compatible if the `cmd` argument is `F_DUPFD`, `F_GETFD`, or `F_SETFD`. BSD has the extra `cmd` arguments `F_GETOWN` and `F_SETOWN` for dealing with job control, which is not applicable under SysV. SysV has the extra `cmd` arguments `F_GETLK`, `F_SETLK` and `F_SETLKW` which deal with file locking (see the section on File Locking).

The `cmd` arguments `F_GETFL` and `F_SETFL` are used in the same way under both systems, but the file status flags being accessed are different. Under SysV, the status values `O_RDONLY`, `O_WRONLY`, `O_RDWR`, `O_NDELAY` and `O_APPEND` may be accessed. Under BSD, `FAPPEND` corresponds directly with `O_APPEND` under SysV, and in fact uses `O_APPEND` as a flag for *open( )* (so you can't just #define it). The BSD flag `FASYNC` is used with job control – there is no equivalent functionality under SysV. The SysV.2 flag `O_SYNC` is used to ensure that data is physically updated on disk after every *write( )*. Under BSD, calling *fsync( )* after every *write( )* can be used to achieve the same effect.

The SysV `O_NDELAY` flag affects reads and writes on pipes and reads from tty devices: a read from an empty pipe or a tty device with no information available, which would otherwise block, returns with 0. The BSD `FNDELAY` flag affects reads and writes on sockets and tty devices: an operation which would otherwise block returns an error and sets `errno` to `EWOULDBLOCK`.

flock(fd, operation) (BSD) See *Locking*
fork( ) (any) *Compatible all versions*
fstat(fd, statbufp) (any) See *stat( )*
fstatfs(fd, buf, len, fstyp) (SysV.3) See *statfs( )*
fsync(fd) (BSD)
The effect of a *write( )* followed by an *fsync( )* can be duplicated under SysV.2 by setting the `O_SYNC` flag on the file descriptor prior to the *write( )*.

ftruncate(fd, length) (BSD) See *truncate( )*
getdents(fd, buf, nbytes) (SysV.3) See *Directory*
This system call is designed to read directory entries in a filesystem independant format. The *directory(3)* routines should generally be used in preference to using this routine directly.

getdtablesize( ) (BSD)
This gives the maximum no. of open file descriptors available to a process. Under SysV, use `NOFILE` defined in `<sys/param.h>`. The information is only available at compile time under SysV.

getgid( ), getegid( ), getuid( ), geteuid( ) (any) See *Access permissions*
The only difference in these functions between versions is the return value. Under SysV, they are int. Under SysV.3, they are unsigned short. Under BSD, they are `gid_t` and `uid_t`, as defined in `<sys/types.h>`.

getgroups(gidsetlen, gidset) (BSD) See *Access permissions*
Used to get group access list under BSD. Under SysV just use *getegid( )* to get the single value group id that is used for all access permission checks.

gethostid( ) (BSD)
Designed to get a unique number (as set by sethostid( )) for networking purposes. Under SysV, read this from a file.

gethostname(name, namelen) (BSD)
Designed to get the standard host name for this machine. Under SysV, this information is provided by the *uname( )* system call.

```
#if    BSD
#   include   <sys/param.h>
    ...
    static char                 hostbuf[MAXHOSTNAMELEN + 1];
```

```
    ...
    hostname = gethostname(hostbuf, sizeof hostbuf) == -1 ? 0 : hostbuf;
#else  /* BSD */
#   include  <sys/utsname.h>
    ...
    static struct utsname  names;
    ...
    hostname = uname(&names) == -1 ? names.nodename : 0;
#endif /* BSD */
```

Under SysV the name of the system (as obtained by *uname( )*) is configured into the kernel and cannot be changed at runtime. An alternate approach to emulating `gethostname( )` and `sethostname( )` is to read and write the name to and from a known file. This approach has the problem that other programs using `uname( )` will obtain possibly different information. It is rare that a user program would need to change the system name except at boot time.

getitimer(which, value) (BSD) See *Time*
getmsg( ) (SysV.3) See *Streams*
getpagesize( ) (BSD)

The value returned by this routine is the system page size. Under SysV this is often defined as `NBPP` (number of bytes per page) in either `<sys/param.h>` or `<sys/immu.h>`, but is rarely required. On BSD it is the granularity of memory allocated by *sbrk( )*.

getpeername( ) (BSD) See *Sockets*
getpgrp( ) (any)

Under BSD, this system calls takes a pid parameter and returns the process group id of the nominated process group. This capability is not available under SysV – the function call does not take any parameters and returns the process group of the current process.

getpid( ) (any) *Compatible all versions*
getppid( ) (any) *Compatible all versions*
getpriority(which, who) (BSD)

Used to get the scheduling priority for a process, process group, or user. SysV can only provide information about the current process.

```
#if    BSD
#   include  <sys/resource.h>
       ...
       errno = 0;
       prio = getpriority(PRIO_PROCESS, 0);
#else  /* BSD */
       errno = 0;
       prio = nice(0);
#endif /* BSD */
```

getrlimit( ) (BSD)

Used to determine what current limits exist on resource usage for the resources process cpu time, maximum file size able to be created, maximum process data area size, maximum process stack size, maximum size core file to be created, maximum resident set size (ie amount of physical memory being used). Similarly *setrlimit( )* is used to set these values. Under BSD both a hard and soft limit may be specified for consumption of these resources. The process may be notified (eg via a signal) when a soft limit is reached, and will be denied the resource when a hard limit is reached.

The only equivalent functionality in SysV is provided by the *ulimit( )* system call, which allows a process to examine and set the file size limit, and to get the maximum possible data area size. There is no soft limit capability.

```
        long    currfilemax, newfilemax, currdatamax;
#if     BSD
#   include   <sys/time.h>
#   include   <sys/resource.h>
        struct rlimit rl;
        ...
        getrlimit(RLIMIT_FSIZE, &rl);
        currfilemax = (long) rl.rlim_max;
        ...
        rl.rlim_max = (int) newfilemax;
        setrlimit(RLIMIT_FSIZE, &rl);
        ...
        getrlimit(RLIMIT_DATA, &rl);
        currdatamax = (long) rl.rlim_max;
#else   /* BSD */
        long    ulimit( );
        ...
        currfilemax = ulimit(1, 0L) * 512;
        ...
        ulimit(2, (newfilemax + 511) / 512);
        ...
        currdatamax = ulimit(3);
#endif /* BSD */
```

getrusage( ) (BSD)
The system call allows a process to enquire about the utilisation of various resources by itself or its children. There is no equivalent functionality available directly through the system call interface under SysV. If such information is required, system accounting may be able to provide it. A process may directly examine the process accounting file, or use the output of the commands *sar(1)*, *sar(1M)*, *timex(1)*, *acctcom(1)*. Probably the most useful approach is to use *timex(1)* to run the command in question and use the resulting output information.

getsockname( ) (BSD) See *Sockets*
getsockopt( ) (BSD) See *Sockets*
gettimeofday(tp, tzp) (BSD) See *Time*
index(str, ch) (BSD)
Equivalent under SysV to *strchr(str, ch)*.

ioctl(fd, request, argp) (any) See *Tty*
The actual use of this command is compatible amongst all versions, though the possible values of the parameters vary widely. This system call is used to perform device specific actions on various devices. The only device where a modicum of standardisation exists is the tty interface.

kill(pid, sig) (any) *Compatible all versions*
killpg(pgrp, sig) (BSD)
This system call can be replaced with the *kill( )* system call under any version.

```
#if     BSD_ONLY
        status = killpg(pgrp, sig);
#else
        status = kill(-pgrp, sig);
#endif
```

link( ) (any) *Compatible all versions*
listen( ) (BSD) See *Sockets*

lseek(fd, offset, whence) (any) *Compatible all versions*

BSD defines tokens for the *whence* argument of this call.

```
#ifdef BSD
#   include  <sys/file.h>
#endif /* BSD */

#ifndef      L_SET
#   define   L_SET  0
#   define   L_INCR 1
#   define   L_XTND 2
#endif L_SET
```

lstat(pathname, statbufp) (BSD) See *Symbolic links, stat( )*

Like stat( ), but provides information about symbolic links, instead of the file referred to by the link. No equivalent in SysV.

mkdir(pathname, mode) (BSD, SysV.3)

This system call was introduced into SysV with release 3. For a non-privileged process, the only alternative is to execute the setuid root *mkdir(1)* command. A privileged process could duplicate the code in *mkdir(1)* to create the directory with *mknod(2)*, but this is not recommended.

```
#if    SysV && SysVver < 3
#include     <fcntl.h>
mkdir(path, mode)
char   *path;
int    mode;
{
    int             pid,
                wstat;
    switch(pid = fork( ))
    {
       case -1:
           return -1;
       case 0:              /* child */
           close(2);
           fd = open("/dev/null", O_WRONLY);
           if (fd != 2)
              fcntl(fd, F_DUPFD, 2);
           execl("/bin/mkdir", "mkdir", path, 0);
           _exit(1);
    }
    /* parent */
    while ((s = wait(&wstat)) != pid && s != -1)
       ;
    if (s != pid)
       return -1;
    if (wstat != 0)
       return -1;
    umask(mask = umask(0));
    mode &= ~mask;
    return chmod(path, mode);
}
#endif
```

mknod( ) (any) *Compatible all versions*
mount( ) (any)
BSD is compatible with SysV up to release 2. SysV.3 is incompatible. The following code handles the simplest case, where the file system being mounted is the same type as the root filesystem.

```
        char    *special, *pathname;
        int     rwflag;
#ifndef         SYSV3
        mount(special, pathname, rwflag);
#else   /* SYSV3 */
        mount(special, pathname, rwflag == 0 ? 0 : 1, 0);
#endif /* SYSV3 */
```

msgctl( ) (SysV) See *Message Queues*
msgget( ) (SysV) See *Message Queues*
msgrcv( ) (SysV) See *Message Queues*
msgsnd( ) (SysV) See *Message Queues*
nice( )
The SysV system call is compatible with the BSD library routine, except that no value is returned by the BSD version.

open(pathname, flags, mode) (any)
For SysV, flag values are defined in `<fcntl.h>`; for BSD they are defined in `<sys/file.h>`. See *fcntl( )* for details on the flag values.

opendir(filename) (BSD, SysV.3, PD) See *Directory*
pause( ) (any) *Compatible all versions*
pipe( ) (any) *Compatible all versions*
plock(op) (SysV)
This system call allows a process to lock (or unlock) its text and data segments in memory. This means that they are immune to all normal swapping. There is no functional equivalent under BSD.

poll( ) (SysV.3) See *Streams*
profil(buff, bufsiz, offset, scale) (any)
These are compatible except for the interpretation of the scale argument. A 1:1 mapping of pc to words in buff is specified by a scale of 0x10000 under BSD and 0xFFFF under SysV.

ptrace( ) (any)
Similar but differences.

```
#if     BSD
#   include   <sys/signal.h>
#   include   <sys/ptrace.h>
#endif /* BSD */

#ifndef         /* PT_TRACE_ME */
#   define    PT_TRACE_ME   0
#   define    PT_READ_I     1
#   define    PT_READ_D     2
#   define    PT_READ_U     3
#   define    PT_WRITE_I    4
#   define    PT_WRITE_D    5
#   define    PT_WRITE_U    6
#   define    PT_CONTINUE   7
#   define    PT_KILL               8
#   define    PT_STEP               9
#endif /* PT_TRACE_ME */
```

putmsg( ) (SysV.3) See *Streams*

quota( ) (BSD)

The *quota( )* and *setquota( )* system calls are used to implement per-user disk quotas on BSD systems. There is no equivalent functionality available under SysV.

read( ) (any) *Compatible all versions*

readlink( ) (BSD) See *Symbolic links*

readdir(dirp) (BSD, SysV.3, PD) See *Directory*

readv(fd, iov, iovcnt) (BSD)

```
struct iovec /* defined on BSD in <sys/uio.h> */
{
    char      *iov_base;
    int             iov_len;
};

readv(fd, iov, iovcnt)
int           fd;
struct iovec  *iov;
int           iovcnt;
{
    struct iovec    *iovt;
    int                   n,
                    nread;
    char            *buf,
                    *bp,
                    *malloc( );

    n = 0;
    /* count up no. bytes to read */
    for (i = 0, iovt = iov; i < iovcnt; i++, iovt++)
       n += iovt->iov_len;
    if (!(buf = malloc((unsigned) n)))
       return -1;
    if ((nread = read(fd, buf, n)) == -1)
    {
       free(buf);
       return -1;
    }
    n = nread;
    bp = buf;
    for (i = 0, iovt = iov; i < iovcnt && n; i++, iovt++)
    {
       int    nx;
       nx = iovt->iov_len;
       if (nx > n)
           nx = n;
       memcpy(iovt->iov_base, bp, nx);
       bp += nx;
       n -= nx;
    }
    free(bp);
    return nread;
}
```

reboot( ) (BSD)
This system call is used to shutdown and possibly reboot the system. It may be sensibly invoked in two possible ways, and these can be emulated under SysV.3 versions.

```
#if    BSD
#   include   <sys/reboot.h>
       ...
       reboot(RB_HALT);     /* halt; no reboot */
       reboot(RB_AUTOBOOT);       /* halt; then reboot multiuser */
#else  /* BSD */
#   include   <sys/uadmin.h>
       ...
       uadmin(A_SHUTDOWN, AD_HALT, 0);   /* immediate halt; no reboot */
       uadmin(A_SHUTDOWN, AD_BOOT, 0);   /* immediate halt; reboot multiuser */
#endif /* BSD */
```

The 2nd parameter of *uadmin( )* could also be A_REBOOT, indicating that the system shutdown is immediate, rather than after killing all user processes, flushing the buffer cache and unmounting the root filesystem. Also note that the 3rd parameter of *uadmin( )* has a machine dependant function.

recv( ) (BSD) See *Sockets*
recvfrom( ) (BSD) See *Sockets*
recvmsg( ) (BSD) See *Sockets*
rename( ) (BSD)
rewinddir(dirp) (BSD, SysV.3, PD) See *Directory*
rindex(str, ch) (BSD)
Equivalent under SysV to *strrchr(str, ch).*

rmdir(pathname) (BSD, SysV.3) See *mkdir( )*
sbrk(incr) (any)
seekdir(dirp, loc) (BSD, SysV.3, PD) See *Directory*
select( ) (BSD) See *Sockets*
semctl( ) (SysV) See *Semaphores*
semget( ) (SysV) See *Semaphores*
semop( ) (SysV) See *Semaphores*
send( ) (BSD) See *Sockets*
sendto( ) (BSD) See *Sockets*
sendmsg( ) (BSD) See *Sockets*
seteuid(euid), setegid(egid) (BSD) See *setreuid( ), setregid( )*
These are library routines on BSD which call the *setreuid( )* and *setregid( )* system calls.

setgroups(ngroups, gidset) (BSD) See *getgroups( ), Access permissions*
Under SysV, use *setgid( ).*

sethostid( ) (BSD) See *gethostid( )*
sethostname( ) (BSD) See *gethostname( )*
setitimer( ) (BSD) See *getitimer( ), Time*
setpgrp( ) (BSD, SysV) See *getpgrp( )*
The SysV call

```
status = setpgrp( );
```

is equivalent to the BSD code

```
status = getpid( );
if (setpgrp(0, status) == -1)
   status = -1;
```

Under BSD, this call can be used to alter the process group of another process. There is no SysV mechanism to do this.

setpriority( ) (BSD) See *getpriority( )*
Used to set the scheduling priority for a process, process group, or user. SysV can only set the priority of the current process.

```
#if    BSD
#   include   <sys/resource.h>
       int    pri;
       ...
       setpriority(PRIO_PROCESS, 0, pri);
#else  /* BSD */
       int    pri, incr;
       incr = pri - nice(0);      /* calculate increment required */
       errno = 0;
       nice(incr);
#endif /* BSD */
```

setquota( ) (BSD) See *quota( )*
setregid(rgid, egid), setreuid(ruid, euid) (BSD) See *Access permissions*
These system calls are used to change the real and effective uid and gid of the current process. If a parameter is given as –1, the current value is used (ie no change is requested). Under SysV, there is no capability for a non super user process to change its real uid or gid, and no capability for a super user process to change its real uid or gid independant of the effective uid or gid.

setrgid(rgid), setruid(ruid) (BSD) See *setreuid( ), setregid( )*
These BSD library routines use the *setreuid( )* and *setregid( )* system calls.

setrlimit(resource, rlp) (BSD) See *getrlimit( )*
setsockopt( ) (BSD) See *getsockopt( ), Sockets*
settimeofday(tp, tzp) (BSD) See *gettimeofday( ), Time*
setuid(uid), setgid(gid) (any) See *Access permissions*
shmat(shmid, shmaddr, shmflg) (SysV) See *Shared Memory*
Attach an already existing shared memory segment to the current process.

shmctl(shmid, cmd, buf) (SysV) See *Shared Memory*
Perform various operations on a shared memory segment.

shmdt(shmaddr) (SysV) See *Shared Memory*
Detach a shared memory segment from the current process.

shmget(key, size, shmflag) (SysV) See *Shared Memory*
Create a shared memory segment.

sigblock(mask) (BSD) See *Signals*
Like *sigsetmask()*, but adds signals to be blocked to the current mask rather than setting them absolutely.

sighold(sig) (SysV.3) See *Signals*
Specifies that the given signal is to be held upon receipt, (BSD "blocked"). Use *sigblock()* or *sigsetmask()* under BSD.

sigignore(sig) (SysV.3) See *Signals*
Used to ignore the specified signal. Pending instances of this sigbals are discarded. Use *sigvec()* under BSD.

sigmask(signum) (BSD) See *Signals*
This is a macro defined in `<signal.h>` on BSD systems which is used to construct a signal mask from a signal number.

signal(sig, func) (any) See *Signals*
This routine is available as a system call under SysV, and as a library routine under BSD. Usage is the same in either case. See the "signals" section for differences in function.

sigpause() (BSD, SysV.3) See *Signals*
This system call is used by both BSD and SysV.3 to unblock (SysV "release") a signal, and then wait for a signal to occur or take action immediately if the specified signal was pending. However, the argument under BSD is a signal mask, used to specify the blocked status of all signals, where under SysV it is an integer specifying a single signal.

sigrelse(sig) (SysV.3) See *Signals*
Specifies that a signal is no longer to be blocked. Any pending signals of this type will take effect. Use *sigsetmask()* under BSD.

sigreturn(scp) (BSD4.3) See *Signals*
Used to "atomically unmask, switch stacks, and return from a signal context". The argument to this routine is a pointer to a signal context, in other words a stack frame. Fiddling with this structure is extremely machine dependant. No equivalent SysV functionality.

sigset(sig, func) (SysV.3) See *Signals*
Used to specify the action to be taken for a given signal: whether it is to be caught, ignored, held, or cause program termination. If a signal handler is specified with this routine it is not affected by the other SysV.3 signal handling system calls.

The *sigvec()* system call provides the equivalent functionality under BSD.

sigsetmask(mask) (BSD) See *Signals*
Specifies which signals are to be blocked (SysV.3 "held"). Use *sighold()* under SysV.3.

sigstack(ss, oss) (BSD) See *Signals*
Used to specify the stack on which signal handler routines will be executed. No equivalent SysV functionality.

sigvec(sig, vec, ovec) (BSD) See *Signals*
Specifies the action taken upon receipt of a signal. The closest equivalent under SysV was *signal()* until SysV.3, where *sigset()* and related functions can be used.

socket( ) (BSD) See *Sockets*
socketpair( ) (BSD) See *Sockets*
stat(pathname, statbufp) (any)
Used in the same manner on any version, but the stat structure varies between SysV and BSD. Fields in common are listed below. The types are defined in `<sys/types.h>` and the actual names may vary from those given here (eg `ushort` in SysV is `u_short` in BSD).

```
dev_t  st_dev;          /* dev on which inode resides */
ino_t  st_ino;          /* inode number */
ushort st_mode;     /* file mode */
dev_t  st_rdev;     /* dev of this inode (if applicable) */
short  st_nlink;    /* no. links */
ushort st_uid;          /* uid of file owner (short on BSD) */
ushort st_gid;          /* gid of file group (short on BSD) */
off_t  st_size;     /* file size in bytes */
time_t st_atime;    /* time of last access */
time_t st_mtime;    /* time of last data modification */
time_t st_ctime;    /* time of last inode modification */
```

Additionally, BSD provides the fields:

```
long   st_blksize;  /* optimal blocksize for i/o */
long   st_blocks;   /* no. blocks actually allocated */
```

statfs(pathname, buf, len, fstyp) (SysV.3)
This system call is used to retrieve generic information about a filesystem. It replaces the *ustat( )* system call used before SysV.3 (though SysV.3 also supports *ustat( )*). Its introduction is because of the filesystem switch introduced with SysV.3 – reading the filesystem superblock directly is no longer a

practicable alternative for the many different possible filesystem types. To obtain information about a filesystem under BSD, or more information than is provided by *ustat( )* under SysV, a process has to open the filesystem directly and read the superblock.

stime(timep) (SysV) See *Time*
This system call is used to set the system's idea of time of day. Under BSD, *settimeofday( )* can be used instead.

strchr(str, ch) (SysV)
This system call is equivalent on BSD to *index(str, ch).*

strrchr(str, ch) (SysV)
This system call is equivalent on BSD to *rindex(str, ch).*

swapon( ) (BSD)
Used under BSD to add a swap device for interleaved paging/swapping. No equivalent functionality under generic SysV. However, some implementations of SysV provide an implementation specific means of doing this. For example on the 3b series running SysV.2.1 and up, the *sys3b( )* system call with *cmd* parameter `S3BSWPI` can be used to add or delete swapping areas.

symlink( ) (BSD) See *Symbolic links*
Used to create a symbolic link to a file. No equivalent in SysV.

sync( ) (any) *Compatible all versions*
sys3b(cmd, arg1, arg2, arg3) (SysV)
This system call is machine dependant. Other implementations of SysV (and indeed BSD) may provide similar machine specific system calls to handle tasks not otherwise provided by the release. Typical uses of this sort of system call are to operate lights on the machine, provide access to internal system tables, access or change kernel configuration parameters, access or change boot information, access or change swap areas, access or change information stored in non-volatile RAM, etc.

syscall( )
Indirect system call. This is very system dependant, and is not intended to be used by C programs.

sysfs( ) (SysV.3)
This system call is used to determine the type of a filesystem.

telldir(dirp) (BSD, SysV.3, PD) See *Directory*
time( ) (any) See *Time*
times( ) (SysV) See *Time*
truncate(path, length) (BSD)
The system calls *truncate( )* and *ftruncate( )* are used to reduce a file to a particular length. Under SysV, trucation of a file is only possible to 0 length.

uadmin( ) (SysV) See *reboot( )*
ulimit( ) (SysV) See *getrlimit( )*
umask(mask) (any) *Compatible all versions*
umount(path) (any) *Compatible all versions*
unlink( ) (any) *Compatible all versions*
ustat( ) (SysV) See *statfs( )*
utime(pathname, times) (SysV) See *utimes( )*
utimes(pathname, tvp) (BSD)
Under SysV, one only needs write permission on a file to change the times. On BSD, only the owner (or super-user) may do this. In both cases, the inode change time is set to the current time, and in neither case can the inode change time be set to any arbitrary value.

```
long   atime, mtime; /* new access and modification times in seconds */
#if    BSD
#   include   <sys/time.h>
       struct timeval tvp[2];
```

```
        ...
        tvp[0].tv_sec = atime;
        tvp[1].tv_sec = mtime
        utimes(file, tvp);
#else   /* BSD */
        struct utimbuf      tbuf;
        ...
        tbuf.actime = atime;
        tbuf.modtime = mtime;
        utime(file, &tbuf);
#endif /* BSD */
```

vfork( ) (BSD)
This system call is provided for efficiency only where the child process intends to immediately exit or call *exec( )*, and can be replaced in SysV with *fork( )*. If you are writing portable code and intend to use *vfork( )* if the code is running on a BSD system, you should read the manual entry carefully before doing so.

vhangup( ) (BSD)
This system call causes a number of actions to take place, basically disassociating the current process's control terminal from any other references.
TODO

wait(statusp) (any)
Although the definition of the parameter under BSD is a pointer to a union wait, rather than to an int under SysV, the usage and meaning is generally compatible.

wait3(status, options, rusage) (BSD)
This system call allows a parent to collect more detailed information about its children, and to optionally avoid hanging.

write(fd, buf, nbytes)	(any)	*Compatible all versions*
writev(fd, iov, iovcnt)	(BSD)	See *readv( )*

4. Include files

BSD and SysV C programs usually include a lot of standard header files. In many cases, the use of these files is equivalent, but very often a file with the same name will have significantly different contents, and in some cases files with similar contents have different names.
TODO

5. curses

System V introduced the terminfo terminal description database scheme to replace the previous termcap database, which is still in use by BSD. At the same time, a new version of the curses library was introduced. Fortunately it is largely compatible with the old version, but there are a number of differences. The most common of these incompatibilities is given below. The best way to write code which distinguishes the two versions is to #ifdef on a token like `A_REVERSE` which is defined in `<curses.h>` in the terminfo version, but not in the termcap version.

Note that BSD programs that use termcap routines only are compiled with -ltermcap, while those that use curses routines must be compiled with -lcurses -ltermcap. All SysV programs that use any of these routines are compiled with -lcurses only.

```
#ifdef A_REVERSE
cbreak( )
nocbreak( )
```

```
#ifndef A_REVERSE
crmode( )
nocrmode( )
```

```
attron(x)
attroff(x)
wattron(win, x)
wattroff(win, x)
beep( )
flash( )
```

```
standout( )
standend( )
wstandout(win)
wstandend(win)
putchar(' 07')
{
    if(BP)
       _puts(VB);
    else
       putchar(' 07');
}
```

```
saveterm( )
resetterm( )
erasechar( )
```

```
savetty( )
resetty( )
ioctl(o, TIOCGETP, &sg), sg.sg_erase
```

6. Directory

The traditional directory structure, and that used in SysV looks like:

```
struct direct
{
        ino_t  d_ino;
        char   d_name[DIRSIZ];
};
```

where `ino_t` is normally a 2 byte quantity and `DIRSIZ` is always 14. This is defined in `<sys/dir.h>` on SysV systems (although it will not be in SysV.4). BSD4.2 introduced long filenames, up to `MAXNAMLEN` (`<sys/dir.h>`) bytes in length. In order to access this new directory structure in a relatively simple and portable fashion, the *directory(3)* library routines were introduced. These should be used wherever directories are to be read. They are in the standard C library in BSD systems, and many vendors provide them for SysV systems, either in the standard C library or in an alternate library (eg try compiling with -lndir). Additionally, there are public domain versions freely available for SysV and other operating systems (such as MS-DOS). Also, SysV.3 has these routines provided as standard.

If the world were a perfect place, this would be all there was to it – just use these routines, and tack a PD implementation of them onto software for systems that don't already have them. Unfortunately, the world not being a perfect place, there is more to the story.

When BSD implemented long directory names, they chose the same structure name in the same include file (`struct direct` in `<sys/dir.h>` to refer to the new directory structure supported by the system, and the `readdir( )` routine returns a pointer to one of these structures. This gives compatibility routines on other systems two alternatives. They could either retain maximum compatibility with BSD, in which case programs using them could not `#include <sys/dir.h>`; or they could opt to change the name of the directory structure returned by *readdir( )*. While routines are available to do the former, the IEEE 1003.1 and the SVID (and various other PD implementations) have opted for the latter approach.

The implication is that you need to handle both BSD-compatible and POSIX-compatible (inc. SysV.3) cases. The following code can be used as a guide:

```
#if    BSD_DIR
#   if        BSD    /* BSD 4.2 */
#      include       <sys/dir.h>
#   endif    /* BSD */
#   if        V9     /* Bell Edition 9, and some others */
#      include       <ndir.h>
#   endif    /* V9 */
#   if        BRL    /* BRL System V emulation */
#      include       <dir.h>
```

```
#   endif     /* BRL */
#   define    dirent direct
#else  /* BSD_DIR */
                    /* SysV.3 or library compatible with same */
#   include   <sys/dirent.h>
#endif /* BSD_DIR */
```

Just in case you thought things weren't too bad, the BSD struct direct is different from the SysV.3 struct dirent. Fields in common are long d_ino (the inode number), short d_reclen (the length of this record rounded up), and d_name (the actual filename, which can be treated as type char * though the actual declaration varies). Most importantly, d_namlen does not appear in the SysV.3 structure, so portable programs should use strlen(...->d_name) instead. Both provide MAXNAMLEN as the maximum possible name length.

That's it as far as most programs are concerned. But I should mention the SysV.3 *getdents( )* system call. It has nothing to do with parking your car in the street, but is designed to read directory entries in a file system independent format. The impetus for this system call is the famous filesystem switch implemented in SysV.3; *getdents( )* can be used to read directory entries regardless of the actual type of the filesystem. The *readdir( )* routine uses this system call and should be used in preference to using this call directly (in fact the manual entry explicitly states that this call should not be used for other purposes).

7. Exec

Note that BSD allows files which begin with "#! interpreter", to be executed directly, while SysV allows only a.out style files to be executed directly. This means that an instance of one of the *exec...( )* routines in a BSD program may have to be changed under SysV to execute the interpreter (typically /bin/sh) for that program instead. For example if `/usr/bin/thing` is a shell script:

```
#if    BSD
       execl("/usr/bin/thing", "thing", "it", 0);
#else  /* BSD */
       execl("/bin/sh", "/usr/bin/thing", "thing", "it", 0);
#endif /* BSD */
```

8. Time

TODO

9. Sockets

Under BSD, sockets are used as the primary IPC mechanism. A socket is described as "an endpoint for communication between processes", with each socket having queues for sending and receiving data. It has been said that sockets are a good idea done badly. BSD has 18 system calls, 25 error numbers and 14 related library routines to implement sockets. Porting a BSD program which uses sockets to SysV will usually require major changes to the code, and probably sigificant design changes. The following list is purely to enable you to identify a BSD program which uses sockets.

System calls: *accept( ), bind( ), connect( ), getpeername( ), getsockname( ), getsockopt( ), listen( ), recv( ), recvfrom( ), recvmsg( ), select( ), send( ), sendmsg( ), sendto( ), setsockopt( ), shutdown( ), socket( ), socketpair( ).* Related library routines: *endhostent( ), endprotoent( ), gethostbyaddr( ), gethostbyname( ), gethostent( ), getprotobyname( ), getprotobynumber( ), getprotoent( ), res_comp( ), res_expand( ), res_init( ), res_mkquery( ), res_send( ), sethostent( ), setprotoent( ).*

10. Tty

TODO

11. Semaphores

TODO

12. Message Queues

TODO

13. Streams

A very nice idea implemented in SysV.3, which would be even more relevant if tty devices had been implemented using streams, which in SysV.3 they are not. A stream is the path between the system call interface to a device (the *stream head*) and the device driver itself. A set of routines, called a *module*, may be interposed in the stream to provide manipulation of data in either direction.

There is no equivalent under BSD.

14. Signals

Using the traditional *signal()* routine, the main difference in behaviour between SysV and BSD is that for most signals under SysV, when a signal is being caught, upon receipt of such a signal the default action is reinstated before the signal handler is called. This means that if another signal is sent to the process before the signal handler has a chance to change things, the process will be killed. This is true for all signals except for `SIGILL` and `SIGTRAP`, for which the action is not reset when the signal is caught. For this reason, many programs use these signals for communication.

Because of this problem, BSD introduced a new set of system calls to handle signals, and made *signal()* a library routine which uses the new system calls. The new *signal()* routine is used in the same way as the SysV system call, except that signal actions are not reset when a signal is caught. The BSD signal routines provide the capability of blocking signals, so that they will be held pending, rather than causing an immediate action. The routines which deal with signals use a signal mask of all the signals currently specified to be blocked. Also provided by BSD is the capability of using an alternate stack frame while executing a signal handler routine. BSD also changed the action of signals which arrive during system calls which are blocked waiting for some event to happen (such as a *read()* on a tty device with no data available). The default action for a signal that has an associated handler routine (ie a signal which is being caught) is for the system call to be interrupted, but then restarted on completion of the handler routine. However it is possible to specify the normal SysV action of having the system call terminate and return an error condition.

The problem of signals is dealt with in SysV.3 by the introduction of a number of system calls to provide similar functionality to that of BSD. Signals may be blocked ("held" in the SysV.3 terminology), released, ignored, or a signal handler routine may be specified. There is no way to set an alternate stack frame, and system calls cannot be made to be restarted – it is up to the programmer to cater for signals or protect critical pieces of code.

15. Symbolic links

BSD supports the concept of symbolic links. A symbolic link is a file in the filesystem which refers to another file. An attempt to open the symbolic link results in a file descriptor referring to the file to which the link is pointing. Similarly a *stat( )* system call and most other system calls which act on files will affect the referenced file rather than the actual symbolic link. An *unlink( )* or *rename( )* system call will affect the symbolic link itself. A new *lstat( )* system call has been introduced to access information about the symbolic link itself.

16. MAUS

This acronym stands for Multiple Access User Space. Some implementations of SysV up to SysV.2 had a set of system calls known under this acronym to implement shared memory. These should normally

be avoided and the more standard (though still only SysV) shared memory operations used instead.

17. Shared Memory

SysV supports shared memory through the use of the *shmget( )*, *shmat( )*, *shmctl( )* and *shmdt( )* system calls.

18. Access permissions

Every BSD process has a set of group ids used for file access permission checking, as well as the real and effective gid used by SysV. The *getgroups( )* and *setgroups( )* system calls are used to access and change this set. When any file access is attempted, the gid on the file is compared with all the gids in the list. Under SysV, group file access is determined solely by the effective group id of the process.

Under SysV, the gid of a file when it is created is set to the effective gid of the process that created it. Under BSD, the gid of the file is set to the gid of the directory in which the file is beaing created.

Under both BSD and SysV, a process may change its effective uid to match its real uid. Under BSD only, a process may also change its real uid to match its effective uid. The same is true of group ids.

Note that when a process has an effective uid set by executing a setuid program, this effective uid is saved seprately from the current effective uid, so that if the effective uid is changed by the process back to the real uid, it may be changed back to the saved effective uid. This is true for both SysV and BSD, and the same applies for group ids.

An undocumented but deliberate feature of SysV only is that a process cannot change its effective uid to its real uid, if that real uid is the super-user.

The sticky bit has a special meaning when applied to a directory in BSD4.3. Where this is the case, in order to unlink or rename a file in that directory, as well as having write permission on the directory a user must own the file or directory (or be the super-user).

OSI into UNIX: the network junkies have a field day

George Michaelson

george@ditmela.oz

Open Systems Interconnection (OSI) is coming. Incomplete, behind schedule, possibly unwanted. Trying to say something sensible about it is almost impossible. The field of comment is littered with many years of blatantly partisan statements from either side, with levels of hype and invective normally associated with anti-IBM hysteria and other socially acceptable forms of relaxation.

Now that OSI based products have finally started to hit the UNIX marketplace, the pressure to draw up plans for migration is overwhelming, although quite why still isn't clear.

1. In search of the roots of a religious war.

Historically, network products have been closed solutions. You used the network your chosen supplier gave you, or designed one in-house. Lots of people thought vaguely about the global network, and talked about "convergence" which was seen as being technically a solved problem, money and politics aside. In reality, there were a large number of highly disjoint networks, and few of them linked to one another, except by magtape.

In 1969 when the US Department of Defence (DoD) funded ARPANET, named after its Advanced Research Projects Agency this picture changed somewhat as a large funding base came along, demanding conformance to such protocols as were developed for its use. Luckily for us, the protocols were designed by "users", and enough people wanted access to force a very diverse bunch of suppliers to toe the line. ARPANET was designed and specified as a "packet switching" network in contrast to "circuit switched" options which up until then was all the PTT's would provide[1]. >From 4 nodes in 1969, ARPANET grew to 40 nodes in 1973, including some reached by satellite (Hawaii and London).

At that stage, divergence between "datagram" and "virtual circuit" camps was in the future. Both are packet switching protocols, but have different implications in terms of resource usage, underlying hardware and their behaviour under load. In 1974 the GPO[2] in the UK implemented EPSS, their Experimental Packet Switching System. This was based on a virtual circuit scheme, in contrast to the ARPA datagram protocol. This was the first PTT provided packet switching service in the world, and lead onto JANET and PSS the academic and commercial X.25 networks in the UK. It would seem from reading papers of the time that the ARPA protocol suite was known, but for "engineering" reasons the virtual-circuit approach was seen as preferable. ARPANET was by then a 40-node network[3] but a public service would have to cope with hundreds of nodes quite soon after launch, and even give a return on investment. At that time, virtual circuit technology seemed to scale better, had more predictable performance and costs.[4] This protocol choice became enshrined in the X-series of standards, for common use

[1] PTT: Postal, Telegraph and Telecommunications bodies An acronym reflecting the blend of French and English as official languages for international cooperation. Since most countries have now separated postal services from telephony and telecommunications, often prior to deregulation and privatisation this acronym is showing its age a little.

[2] General Post Office: the joys of pre-privatisation days when postal services and telephony were one happy family...

[3] it's interesting to see that 2 PDP-1's 16 PDP-10's and 4 PDP-11's were on the network, with 4 IBM360's and 2 IBM370's. From small acorns mighty forests grow.

[4] Even with hindsight, this choice wasn't so stupid. Although IP nets have been reworked to cope with thousands of nodes, there have been periods when the network was "dead" with end-to-end delays exceeding 30 seconds per-character

by all PTT's.

Having started along the path of providing X.25 aligned services, further divergence seems to have been rapid and inevitable. The PTT's were constrained to cooperate through meetings of the International Telegraph and Telephone Consultative Committee, (CCITT) a UN recognised umbrella organisation with cumbersome paperwork and procedural overheads. Adoption of a flexible approach to upper protocols and application specs by the CCITT was not plausible. Adoption of a rigid structure by the ARPA community was equally unlikely.

Even as late as 1978 it seemed to be assumed applications would sit directly on the transport protocol, as they do in many of the Internet Protocol (IP) based applications -since transport layer was providing end-to-end connectivity on top of the network, what remained to be resolved was arguably for private agreement within the application itself. Had this remained true, divergence would have been much less serious.

At this stage, OSI was still only a reference point, a basis for comparison. Subsequent developments to produce a concrete specification aligned to it, materialising in the 1983 International Standards Organisation (ISO) OSI reference model formalised the rift between the ARPA and the non-ARPA worlds, the latter often viewing ISO conformance as a "must", even if alternative network standards could offer higher functionality. In fact ARPA never restricted itself to a single model for its network, and different "layerings" can be imposed for different functionality, quite apart from the various routing, gateway, relay and other "special" protocols developed to keep it running.

In 1980 Transmission Control Protocol (TCP) was adopted by ARPANET, providing reliable end-to-end transport-level connectivity. TCP supports directly the current stable of applications we all know and love. The DoD paid BBN[5] to develop, and Berkeley to port TCP/IP into UNIX, and the rest is [4.2BSD] history.[6] Of course ARPA, latterly DARPA is no longer the single most important element, but amongst the MILnet, NASA, NFSnet and the various regional backbones hereafter referred to collectively as the (DARPA) internet, IP based protocols still predominate.

2. Why should I care about OSI?

"Intent to migrate" is now commonly accepted in the internet itself and government aligned groups. Anyone in receipt of US federal funding must now adopt an OSI conforming tendering process. Until this year, this was all in the future, but with stabilisation of much of the OSI standards, and the arrival of actual OSI-based systems in certain areas, real changes are starting to show in the IP camp. That isn't a good reason for following suit, but it shows which way the big money points.

Much of the desire to shift reflects a change in R & D funding in general. The network-at-large now comprises several diverse coalescences of common interest groups, and similarly disparate funding bodies, some funding a regional network, others national or international but not for all traffic. The community has expanded from academic/research only to include substantial commercial elements, and crosstalk between the internet and more public networks like ATTMAIL and Tymnet grows steadily, if in restricted contexts. As the community grows, so does its need for "nasty" side-issues like charging and access control software in the underlying code. Bolting this into an existing protocol suite is almost as much effort as adopting a new one which (hopefully) has it built-in. That speed and functionality is lost counts for less in the eyes of the paymasters.

Another argument for adoption of OSI protocols is that newer network technology such as ISDN can only be used under OSI protocols but that's obviously hype. There might be functionality IP-based systems would never exploit, but the existing applications could certainly be used. Politically however it is unlikely major porting effort will be carried out, since a financial commitment to OSI has already

---

echo. The scalability of IP is there, but the performance degradation can be extreme without juggling of internal links, and massive increases in bandwidth. The PTT usually provides the IP network backbones, and clearly they have resources available, but you can also see difficulties in trying to maintain a national network of this type.

[5] A major contractor for ARPA, and historically very closely involved in its development.

[6] Some of us tend to forget that there is life outside of BSD, and SysV based systems have lacked IP support until very recently. Do most pro-OSI UNIX people use SysV instead of BSD?

been made. That doesn't mean IP won't be carried, but newer applications that exploit non-IP based services may not have a backwards compatible IP path.

A better argument is that OSI by design bridges the local/public network gap, through network relaying. Certainly many of the "seven layer cake" diagrams show this, but in practice higher level relaying has to be carried out. Also, IP is easily "tunneled" over X.25 or serial lines, and several packages for this are now available for UNIX, VMS and other systems. This doesn't have quite the same functionality, since arbitrary nodes cannot talk into your applications unless on that specific IP-link.

3. Will IP and OSI integrate cleanly?

In a word, no.

Aside from the diverged layering of their respective models, current OSI offerings do not integrate cleanly into the existing utilities, nor into underlying models of IP subnetting and the like. Nobody appears to be attempting to take existing TCP products and make them work over OSI transport class 4, at least nobody is owning up to it -so you can expect to have to support several competing toolsets, and probably two competing address and hostfile formats for quite some time. The possible exception to this is for BSD users, of which more later.

4. Whats available now ?

End-users don't care very much how it's done, they just want to have it on a plate. Most of them will be aware of and use four core (Mail, News, Remote login access, File transfer) and two "luxury" (remote execution, and network filestore) applications. The ordering of these requirements is purely subjective. People with logins on different hosts tend to want to actually work "at" them, rather than specifying everything locally through rsh commands. Similarly, NFS although enormously useful isn't strictly speaking essential, nor available across all UNIX systems whilst some form of news mail and remote login (cu, tip, telnet, pad) is much more common.

Were a common and consistent mechanism for files-in-mail available, nobody would really care about ftp, but the techniques of sending source are so diverse, and ftp, so simple to use, that it ranks high in most users minds. internet users would certainly complain if anonymous ftp was withdrawn overnight.

Desirable Applications and their OSI-availability

Application	"common OSI names"	status	availability/comments
Mail	X.400,MHS	IS	medium: 3rd party, some UNIXes at extra cost, in 4.4BSD
News	No Equivalent		can be kludged into frameworks, may be done over X.400
Remote Login	VT (Virtual Terminal)	DIS	low: 3rd party in development, may be in 4.4BSD
File Transfer	FTAM (File Transfer Access & Management	IS	medium: in 4.4BSD
Remote Execution	JTM (Job Transfer & manipulation)	IS?	low: not in 4.4BSD
Network Filestore	No Equivalent		cuts across several standards, FFS

note

IS means the standard is ratified and adopted within the ISO/CCITT community as an International Standard. That doesn't mean it never changes, but it does mean the changes are regular (4 yearly cycle) and can be planned for. DIS means it is Draft IS, and highly likely to become IS, although things can stay DIS for a very long time, and change massively when they become IS. IS? means I don't really know anything about this and am fudging the issues.

As you can see some of your needs will be met quite rapidly, if you take the standard Berkeley distribution. Mike Karels announced at AUUG that 4.4BSD will adopt ISODE, along with sockets code for Transport and OSI Network from elsewhere[7] which is where most of this will come from. ISODE by then will encompass VT as well as it's existing FTAM application. The PP mail system from UCL and Nottingham may also be in the 4.4BSD tape, and is being developed using ISODE so the release will have some self-consistency as well as being reasonably complete.

If Berkeley have resolved the differences between IP and OSI network addressing, low-level bridging and relaying between them may be built-in. If not, application-level "fudges" may have to be done. It would be very nice if end-users were not aware of any difference between IP and OSI networks.

5. How about non 4.4BSD systems?

Alas outside of Berkeley, few suppliers are ready and able to offer OSI tools now. Some have OSI Mail systems, because this is the first PTT provided OSI service. AT & T are a major OSI mail service provider, and you would expect them to have code for SysV which will tie into the "ATTMAIL" offering but I have seen no announcements about this. Typically the mail offerings use the existing user-interface (UIP in OSI-speak) and patch into it via a sendmail submission process. This requires users to specify X.400 addresses in the RFC822 To: format, and can be a bit clumsy. The packages are also rarely cheap, and some won't talk to all of the others. Where suppliers do offer OSI network code, few have integrated this cleanly with the existing IP products, and some even demand extra hardware to do OSI network or link layer.

Unless you get 4.4BSD your ability to use these tools is in doubt. Your supplier will probably view them as value-added products and demand high payment for them as a return on their development

[7] I think he said Wisconsin but I was listening so hard I forgot to take notes

costs. If IP-based tools were only available under the same situation, I suspect most BSD-aligned users would migrate to "pure" BSD rapidly, and force a re-think by the suppliers. When OSI becomes essential for your users, market pressures will force suppliers to re-bundle the OSI network offering if they are to win the tenders[8] so things should get better as the market expands.

6. What about non-OSI applications like news and NFS?

The lack of standardisation for news and network filestore will cause at lot of pain. News is probably going to be implemented over X.400 as a private "document structure" by several groups, and end-user tools will remain much the same, but this is a real kludge. Since X.400 mail encompasses higher reliability, and in it's 1988 standard[9] has Distribution-List control software, this should make mail-lists somewhat more acceptable. Some pressure exists for OSI to develop news-aware standards but this is grassroots, and not likely to succeed.

Network Filestore may not be addressed for some time. Some proponents argue the FTAM protocols should be used to transfer the information, whilst existing implementations (eg NFS) deliberately reduce protocol overheads to increase speed. Since FTAM lives in the application layer, above 6 distinct and complex functional layers, the underlying service implementations will have to be massively optimised before this will be plausible. Hopefully a fresh standard area will address this directly rather than trying to cram the concept into the existing model.

7. Anything good to be said for OSI?

Basically, OSI can't yet offer a consistent, integrated view of network access to rival the BSD rsh, rcp, rlogin model. FTAM can and will be directly comparable to ARPA ftp, whilst X.400 mail could potentially blow email users minds out of the water, when its ability to handle mixed text, binary, voice and graphical data is fully extended to them. Existing systems tend to use the X.400 protocols for text-only transfer. Similarly VT will align quite favourably with telnet and the X.29 PAD programs some of us already use, and should offer more screen oriented styles as well, hopefully not just to plug into IBM applications.

For mail and ftp, gateway code into OSI is already in existence. The mail code is used every day for relaying to the UK, and can also be used to reach OTC connected X400 services. Most of you can get to the same people faster via the internet so until features like FAX and TELEX gateways go public this may not be so useful.[10] For VT, telnet-VT gateways are in development. This means total network connectivity for the 3 core functions will rise massively, as the OSI implementations should bridge the current gap between local networks and X.25 or similar WAN access, much more than point-to-point IP tunnels can achieve.[11]

Local users with a mix of OSI and IP hosts and applications will have the worst of both worlds, having to maintain dual sets of almost everything.

Users of VMS and VM will find life gets slightly easier, since both manufacturers will provide built-in gateways to OSI, and so the various 3rd party options used at present may not be needed. In parallel of course they are finally providing TCP/IP support embedded into their systems so you can also choose to ignore OSI for some time to come.

---

[8] In the UK, where central funding provides most academic equipment, contracts and tenders are already beginning to demand a commitment to OSI migration, as in the US. One hopes the funding cover the costs. Previous experience with JANET requirements suggests this sort of planned bullying of suppliers is very effective in getting things bundled: -no OSI network support, no purchase...

[9] which won't be available as implementations for some time yet

[10] This will instantly make over 15 Million people or organisations visible to you, which is a sizable increase in connectivity by most peoples standards.

[11] The first places these get put is between existing modem linked hosts so total connectivity doesn't always rise, although functionality does.

8. It'll be solved in the Directory Service

Many OSI applications pay lipservice to the need for directory services. That these are not in place, nor will be for some time to come doesn't seem to cause much worry, but if too many OSI applications become installed using ad-hoc measures, your worst fears about migration will come true. Until something sensible arrives, tailoring files and host-lists and conversion tables will mushroom in both size and number. Somebody will hack extensions into bind and similar systems, but this may not actually be sufficient. You'll find that many things can be done, but only by the few people with access to the knowledge. Next time somebody near you says "it'll be solved in the directory service" ask them when they expect to get one running.

9. Why Bother migrating now?

In practice, you can sidestep most of the hassle by holding back until US and European market pressure forces manufacturers to things properly. If you have to be "leading edge" then you'd suffer pains willingly, if all you want is enhanced connectability, you may be able to avoid total conversion for some time, and use application specific relay and bridging software.

Systems developers have a nightmare, since they will be finding OSI mandated in tenders, and thus be forced to implement possibly before things get stable. To mimic 4.4BSD functionality you could consider getting ISODE.

The problem OSI was designed to solve has almost solved itself, by the traditional process of de-facto standardisation and market choices. Unix, in sweeping the board has reduced many problems down to re-porting commonly available solutions to *your* flavour of box, if it hasn't already been done. Interworking between SysV and BSD has been almost completely resolved as regards IP-based packages. The otherwise regrettable domination of the PC market by IBM has also lead to a reduction in the inter-connection problem. So if you can tolerate existing ad-hoc approaches, stick with them for another year or so and review the situation.

If like me, your subsistence depends on this confusion, have no fear. OSI isn't going to make network hackers redundant for a long long time.

ISO/OSI Networking Protocols under Berkeley UNIX

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

ISO/OSI Networking Protocols: Project Overview

OSI protocols have enough layers, options to keep several groups busy!

Collaborative project with funding from NBS (US National Bureau of Standards) for OSI protocols in POSIX framework:

- Marshall Rose (Northrop/Wollengong)– ISO Development Environment (session, presentation, FTAM, etc.)
- University of Wisconsin– transport and network
- University of California, Berkeley– socket updates, kernel integration, link layer; POSIX interface
- University College London, University of Nottingham– X.400
- Mitre– Virtual Terminal Protocol
- NBS– Directory Services

University of Wisconsin contributions

Implementations of:

- Transport (TP-4; maybe TP-0/X.25)
- Connectionless network protocol (CLNP) over Ethernet and X.25 (with CONS/COSNS)
- End System-Intermediate System Protocol (ES-IS) (implemented in daemon process)

UC Berkeley work

- updates to socket interface
- kernel integration
- link layer (802.3/802.2, X.25)
- POSIX interface
- POSIX networking extensions?

Problems in integration of OSI protocols into 4.3BSD

- 4.2/4.3BSD offer network-independent socket interface with multiple protocol families, address families.

- Interface supports TCP/IP naturally, supports protocol features and layering similar to TCP/IP.

- Protocol features *not* supported gracefully by 4.3BSD:
  - long network addresses (>14 bytes)
  - multi-level routing hierarchy
  - server confirmation of incoming connections
  - Receipt of lower-level protocol data with user data (or connection request)
  - record marks

Changes in socket interface

- Variable-sized addresses:
  - generic address structure (sockaddr) has length field
  - generic kernel sockaddrs are allocated dynamically (routing layer)
  - Source-, binary-compatible

- Server acceptance of incoming connections:
  - Protocol may pass descriptor to server before confirming connection using *accept*().
  - If server does *read* or *write*, connection is automatically confirmed.
  - If connection is closed immediately, connection is denied.
  - Server may receive user call data or interrogate lower protocol layers.

Changes in socket interface

- Receipt of protocol information, status with data (recvmsg):
  - New *recvmsg* call, new *msghdr* structure.
  - Second buffer pointer and length for typed control information.
  - Received-data flags include end-of-record, datagram-truncated, control-data-truncated.
  - User may pass control buffer or end-of-record flags to protocol with *sendmsg*.
- *send*, *recv* calls deprecated (use *sendto*, *recvfrom*).

Changes in routing layer

- Generic routes must include variable-length destination, gateway addresses.
- Routing ioctl commands to be replaced with message-oriented routing socket.
- Per-address-family network comparison functions replaced with network-part bit mask (byte length plus last-byte mask).
- Network plus host route tables replaced with single hierarchical table.

Remaining problem areas

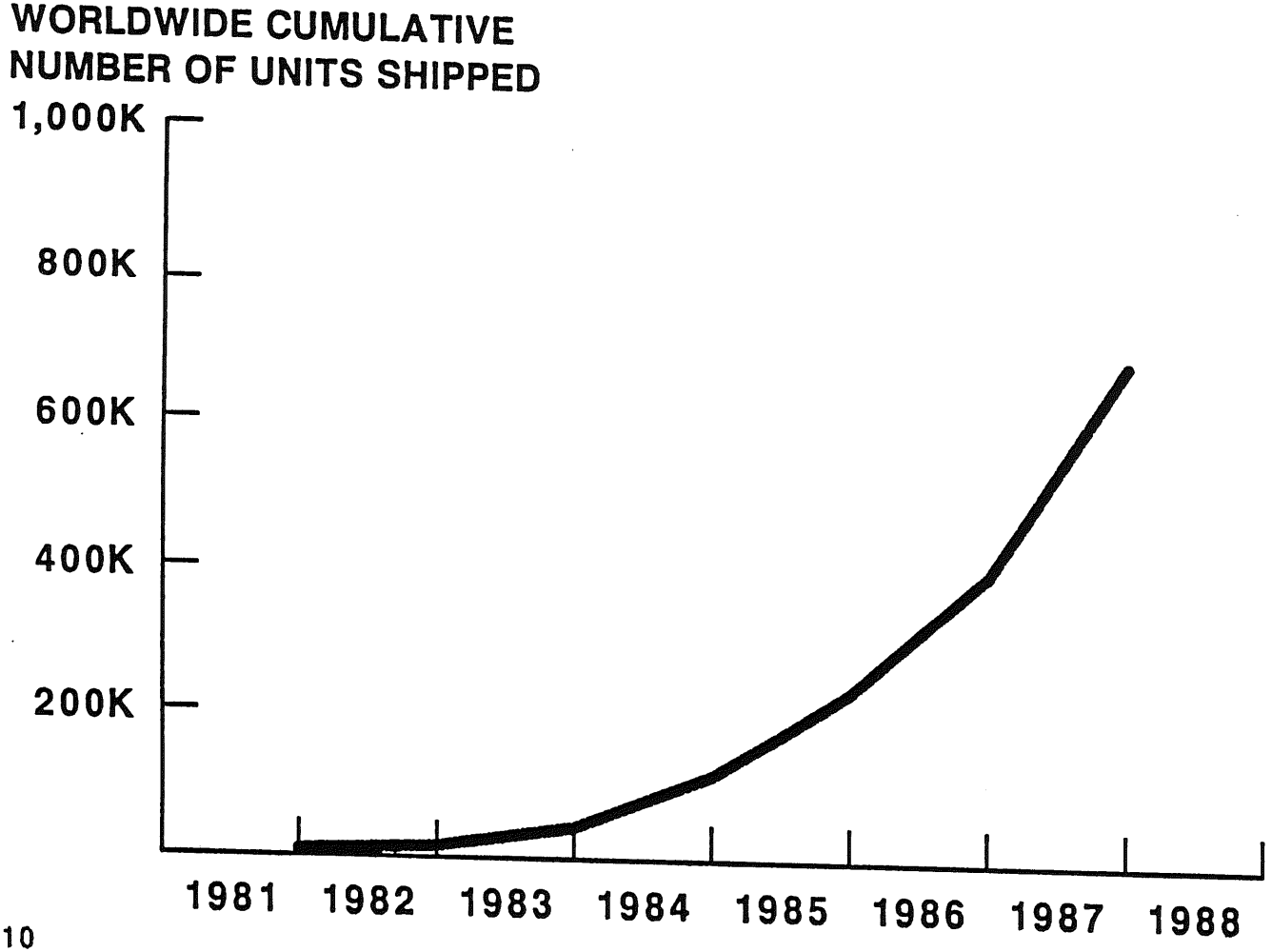
- Connection-oriented interfaces (X.25):
  - Current link-layer interface implies packet-switched network.
  - CLNP over X.25 is easy.
  - Connection-oriented link layer may need streams-like interface (TP-0/CONS/X.25).
- Fill-on-demand routing tables
- "Raw" interface to link layer (IS-IS over 802.2, X.25)

Larry L. Crume
AT&T Unix Pacific Co., Ltd.
AT&T Japan Data Systems Group
Tokyo, Japan

THE FUTURE OF UNIX* SOFTWARE
THE OPEN COMPUTING PLATFORM FOR 1990's

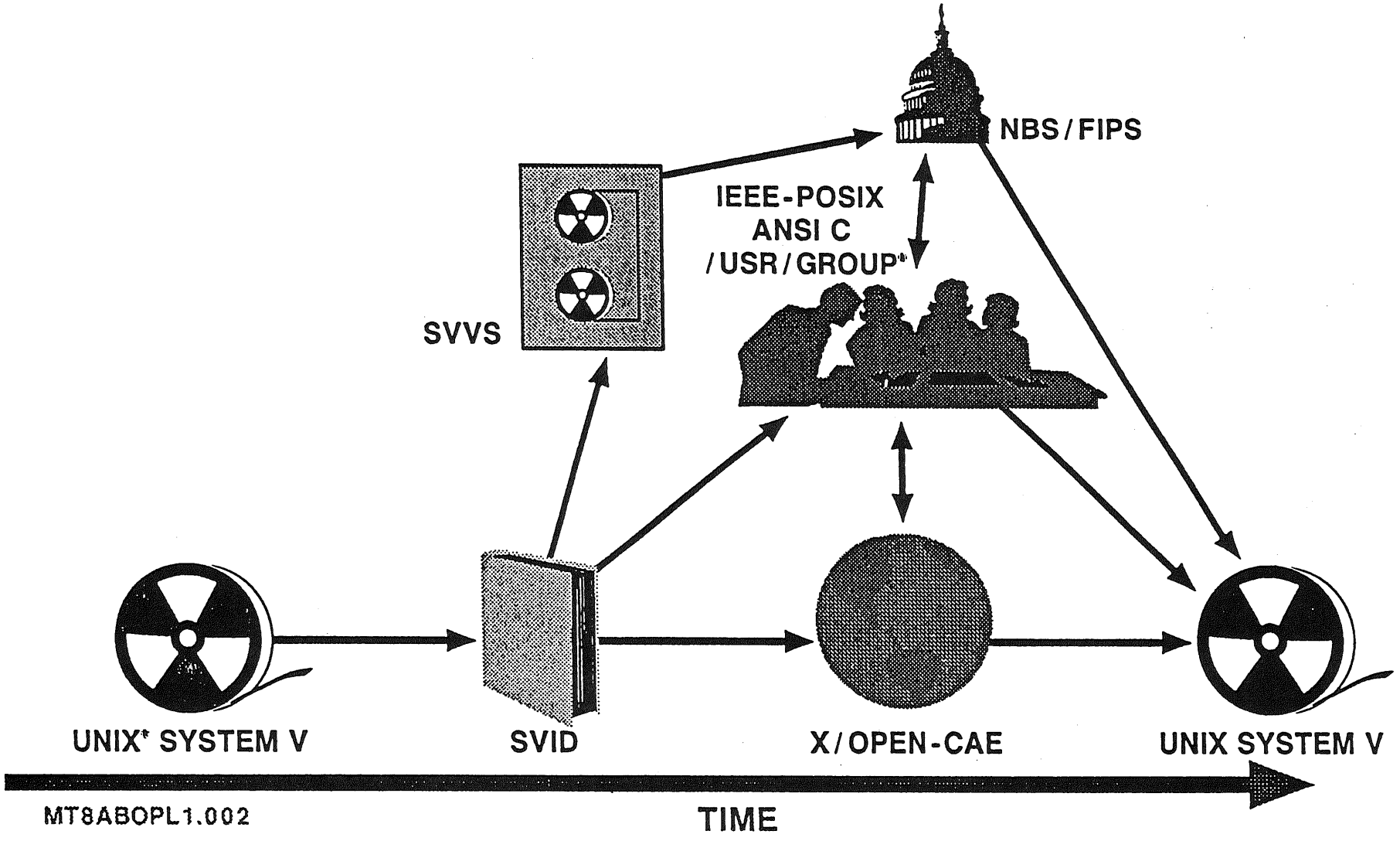
* Registered trademark of AT&T in the USA and other countries.

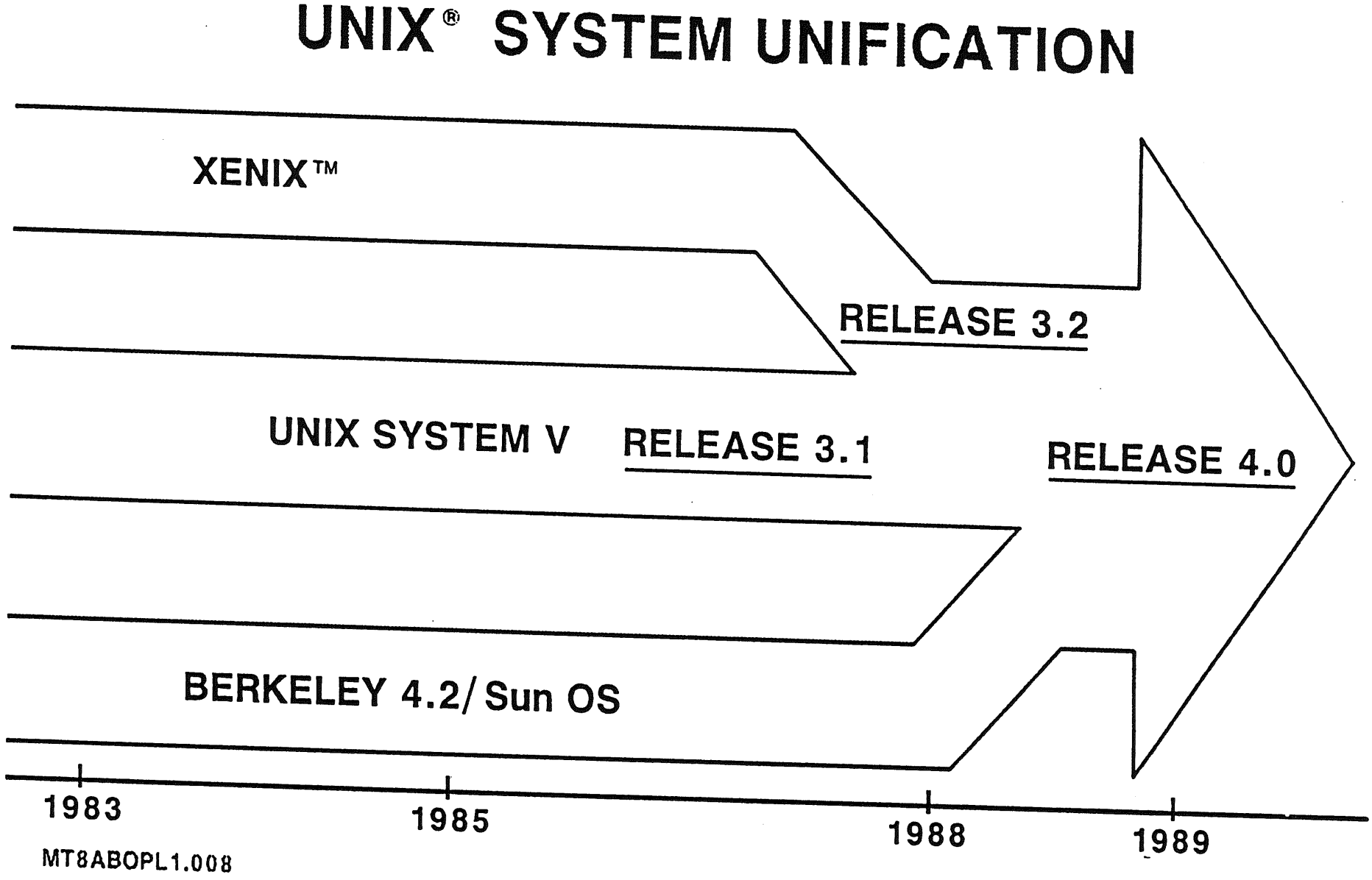
UNIX® SYSTEM MARKET GROWTH



MT8ABOPL1.010

SYSTEM V STANDARDIZATION




UNIX® SYSTEM UNIFICATION
XENIX™
RELEASE 3.2
UNIX SYSTEM V
RELEASE 3.1
RELEASE 4.0
BERKELEY 4.2/ Sun OS
1983
1985
1988
1989
MT8ABOPL1.008

UNIX® SYSTEM UNIFICATION
XENIX S III
XENIX S V
XENIX
XENIX Sun OS BSD
BSD
Sun OS 4.0
Sun OS 4.1
SYSTEM V RELEASE 3.0
SYSTEM V RELEASE 3.1
SYSTEM V RELEASE 3.2
SYSTEM V RELEASE 4.0
MT8ABOPL1003

UNIX® SYSTEM V
1987 Product Activities

- 3B2 porting base
  - Release 3.1
- 386 port
  - Release 3.0
- Internationalization
  - Japanese
  - Korean
  - French
  - German

MT8ABOPL1.005

UNIX® SYSTEM V
1987 Unification Activities

- Microsoft agreement
  - XENIX® consolidation
  - Establish Intel standard ABI
  - Trademark licensing
- Sun agreement
  - BSD, Sun OS™ consolidation
  - Establish SPARC™ standard ABI
  - Trademark licensing
- Endorsements
  - IEEE POSIX
  - ANSI C
  - NBS
  - X/OPEN

MT8ABOPL1.006

SYSTEM V

Application Binary Interface (ABI)

- SVID
  - Defines source code interfaces
  - Provides application source code portability across architectures
- ABI
  - Defines binary interfaces
  - Provides application binary portability within an architecture

    e.g:
    - SPARC™
    - Intel
    - Others
- Benefits
  - PC style application portability
  - Larger software market
  - Preservation of customer investments

UNIX® SYSTEM V
Applications Binary Interface (ABI)

We are working with other major (microprocessor) processor technology vendors to support ABIs for additional architectures:

- **INTEL® 80386**
- **Intergraph® Clipper® Architecture**
- **MIPS® R3000**
- **Motorola® 68XXX and 88XXX**
- **SUN Microsystems® SPARC™**

UNIX® SYSTEM V
Applications Binary Interface (ABI)

The following companies have licensed the SPARC™ technology from SUN Microsystems®:

- **XEROX®**
- **UNISYS®**
- **ICL®**

These companies can utilize AT&T's ABI for SUN Microsystem's RISC architecture (SPARC)

MT8ATR421.005

UNIX® SYSTEM V
1988 Product Activities

- Deliver
  - 3B2 Release 3.2
  - 386 Release 3.2
  - JAE 2.1
  - Japanese Messaging System
  - JAE/386

MT8ABOPL1.007

UNIX® SYSTEM V
Release 4.0 Directions

- Operations, administration and maintenance
- Real time
- Internationalization
- POSIX conformance
- X/OPEN capabilities
- XENIX® / BSD consolidation

MT8ABOPL1.021

UNIX® SYSTEM V
Release 4.0 Future Directions

New Features

- Operations, administration and maintenance
  - Backup and restore
  - Configuration management, s/w installation and distribution
  - Message handling facility
  - Remote operation
- Realtime
  - Scheduler
  - Asynch I/O
- Internationalization
  - ANSI C
  - Message handling facility

UNIX® SYSTEM V
Release 4.0 Directions

Unification

- BSD
  - Signals
- Sun OS™
  - NFS
  - RPC
  - X/NeWS™
- XENIX®

MT8ABOPL1.020

UNIX® SYSTEM V
1988 Unification Activities

- Implement POSIX conformance
- Deliver XENIX® consolidation
- Develop BSD / Sun OS™ consolidation
- Implement X / OPEN capabilities
- Industry Review Draft SVID
- Industry Review Draft SPARC™ ABI

MT8ABOPL1.016

AT&T - SUN ACTIVITIES

Phase 1 - Sun 4 Systems

- Sun development / AT&T consultation
- SVID compliance of Sun OS™ 4.1

Phase 2 - UNIX® System V Release 4.0

- AT&T development / Sun subcontractor
- POSIX compliance
- Berkeley merge
- New AT&T features

Phase 3 - UNIX System V Release 5.0

- AT&T development / joint exploration
- Kernel restructure

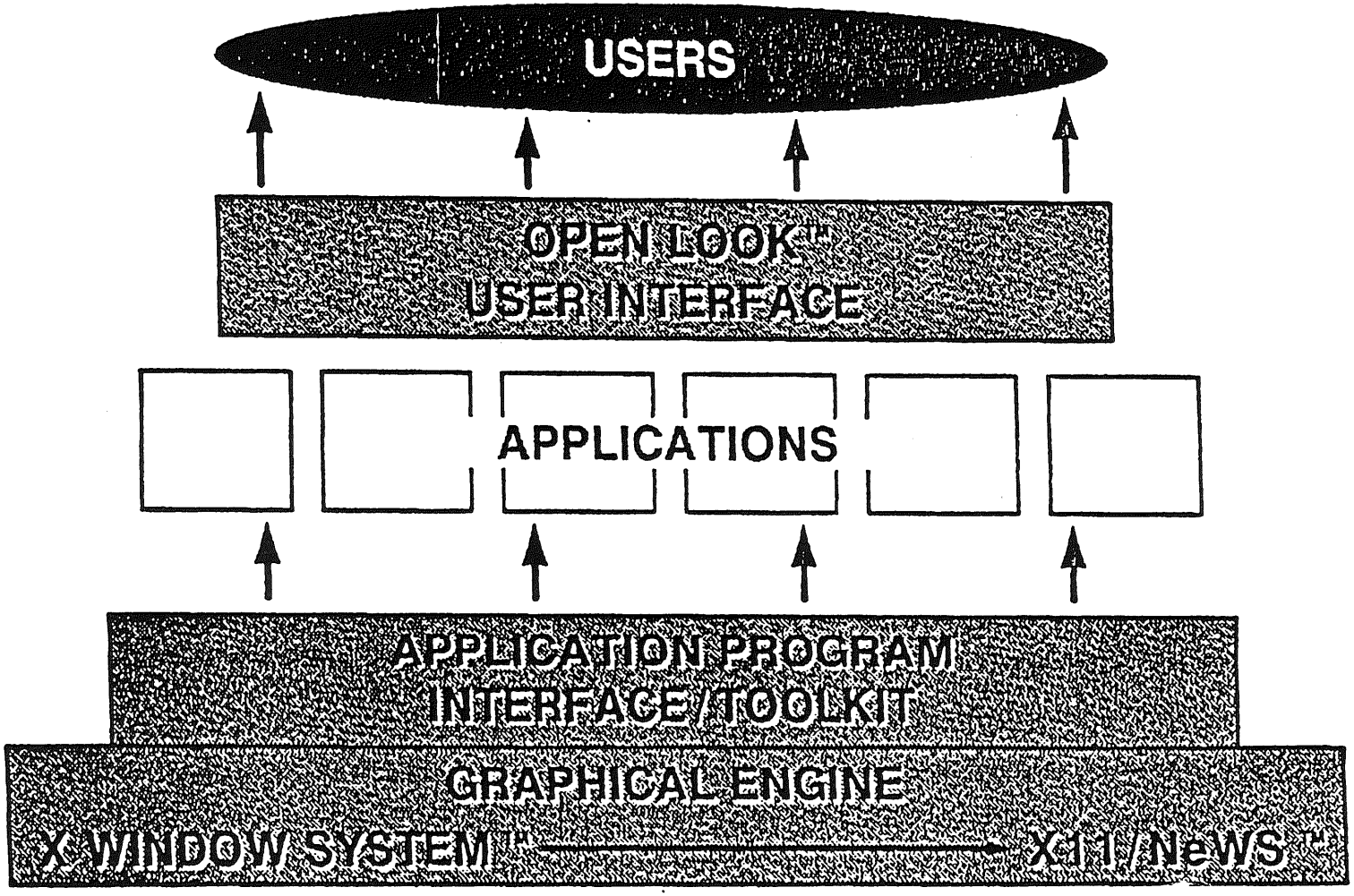
CA5761.117

☐ KEF

THE OPEN LOOK™ GRAPHICAL USER INTERFACE

OPEN LOOK Is A Trademark Of AT&T

OPEN LOOK™ USER INTERFACE



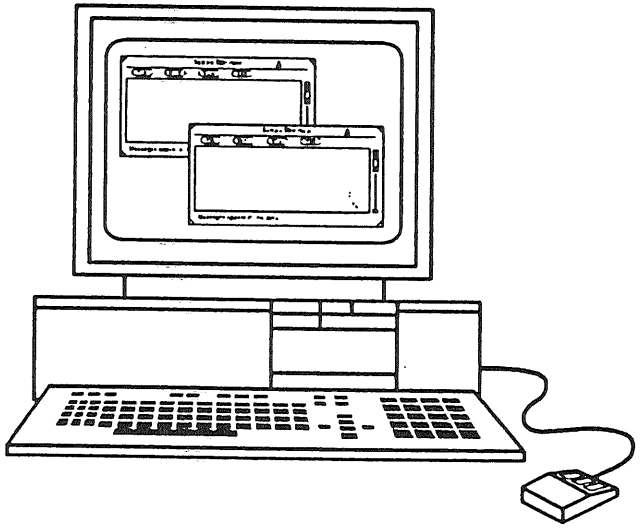
MT8ATR421.007

CA7893.105

LJC

WHAT IS A GRAPHICAL WINDOWING SYSTEM?

- A Collection Of Programs Which:
  - Allow For Multiple Applications To Appear Simultaneously On A Display Device
  - Support Interactive Graphics And Pointing Devices



Vol 9 No 5

OPEN LOOK™ GRAPHICAL USER INTERFACE

A STANDARD "LOOK AND FEEL"

- AT&T Offers "Look And Feel" To The Industry
- OPEN LOOK Specification Published

CA7893.116 BR-FRG

WHAT IS A STANDARD LOOK AND FEEL?

- A Consistent Method Of:
  - Displaying Information To The Users (Look)
  - Allowing The User To Interact With The System (Feel)
- Interface Levels
  - End User: Standard Method Of Communication Between The System And End-User (OPEN LOOK™)
  - Programmer: Programmer Toolkit (API) To Develop Applications That Adhere To The Standard Look And Feel (The OPEN LOOK™ Xt Toolkit)

Vol 9 No 5

OBJECTIVE AND STRATEGY

OBJECTIVE

- Provide An Industry Standard Graphical Windowing System And User Interface For UNIX® System V

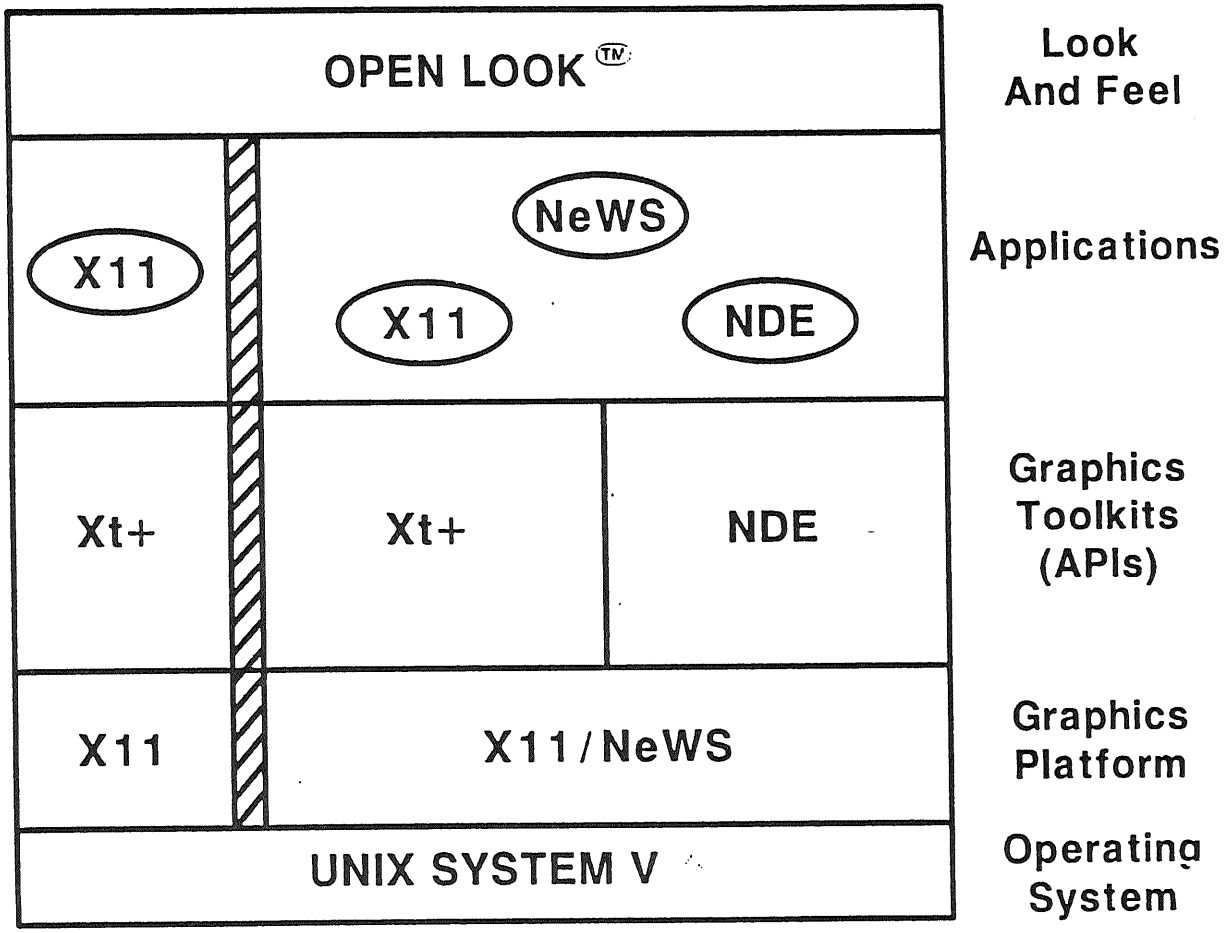
STRATEGY

- Stabilize Character Graphics Software
- Adopt The X Window System And Implement For UNIX System V
- Migrate To X11/NeWS On UNIX System V
- Define And Implement A Standard UNIX System Look And Feel (OPEN LOOK™)
- Provide Graphics Toolkits (APIs) For Application Developers

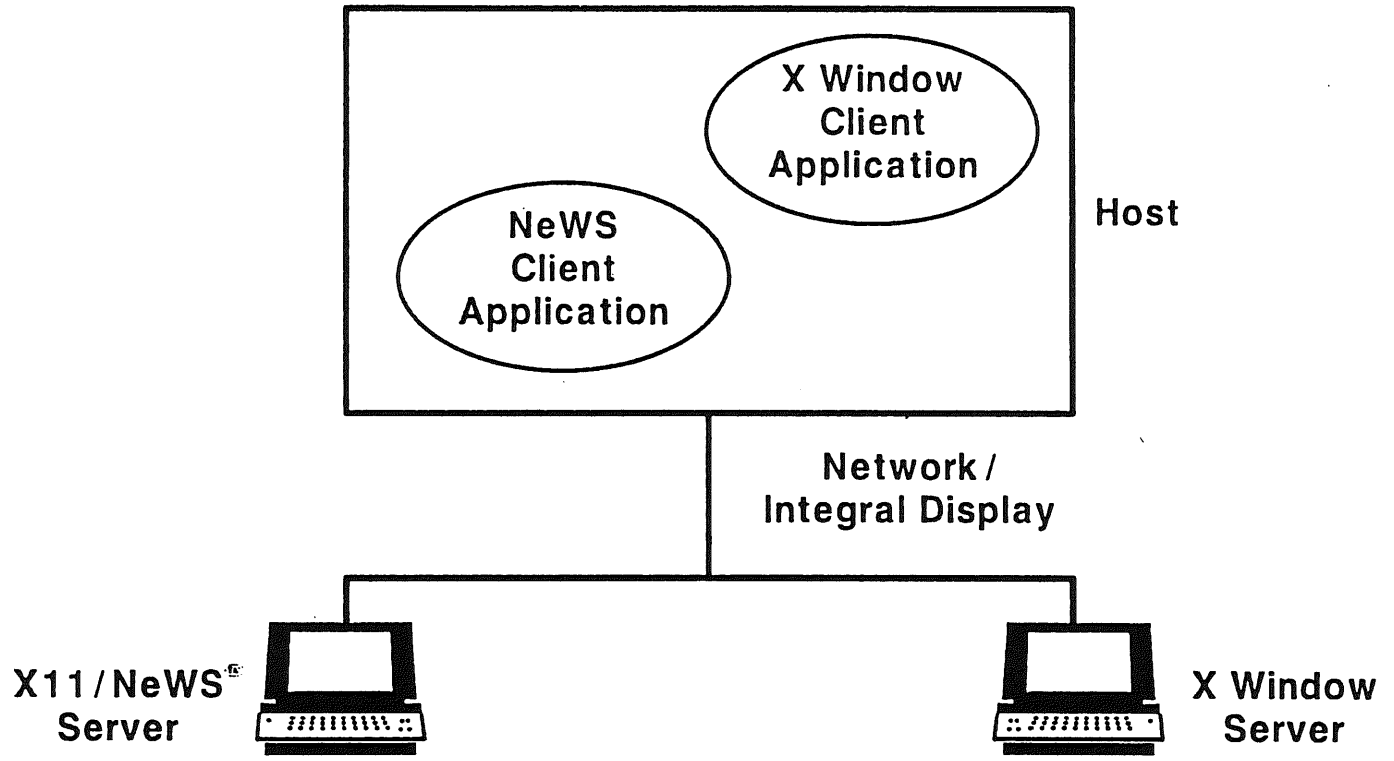
CA7893.145

☐ BR/BAD

UNIX® SYSTEM GRAPHICS USER INTERFACE STRATEGY



X11/NeWS GRAPHICS SOFTWARE



- Applications Must Be Written To Either X Window Or NeWS Graphics Standard. Target Display Devices Need To Be A Factor In Deciding Which Standard To Use
- Graphics Servers Manage Displays
- Any Client Application Can Connect To A Local Or Remote Server Of The Same Type (X Or NeWS)

API TOOLKITS

- Object Oriented Programming Environment
  - Programmers Link Pre-defined User Interface Objects Into Their Application
- Toolkit Objects Include:
  - Scroll Bars
  - Buttons
  - Menus
  - Pop-up Windows
  - Notices
  - Context Sensitive Help
- Two Toolkits
  - Xt+
  - NDE

CA5761.107

☐ KEF

OPEN LOOK™ GRAPHICAL USER INTERFACE

BUILD OPEN STANDARD

- Expand UNIX® System V Market Place
- Create Recognized System V User Interface
- Create Industry Standard "Look And Feel"
- Industry Input Program
- License Trademark
- License Source Code

CA5761.108

☐ KEF

OPEN LOOK™ GRAPHICAL USER INTERFACE

INDUSTRY INPUT PROGRAM

- Solicit Industry Comment
- Keep The Industry Informed
- Circulate Draft Spec — July 1988
- Circulate Style Guide Draft — September 1988
- Circulate API Drafts — September 1988

Vol 9 No 5

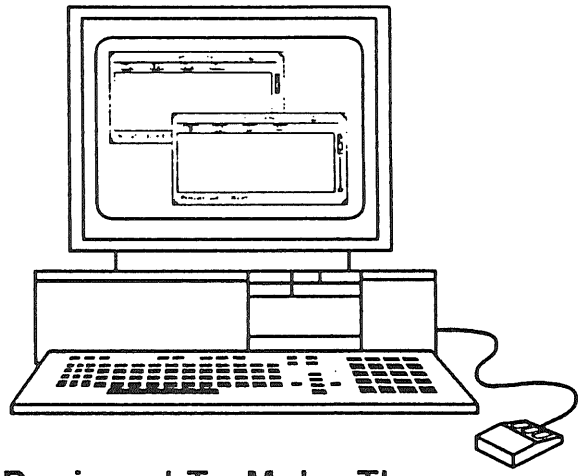
OPEN LOOK™ GRAPHICAL USER INTERFACE

TRADEMARK LICENSING PROGRAM

- Encourage Licensing Of The OPEN LOOK Trademark
- Requires Compliance With Specification
- Maintain A Consistent "Look And Feel"
- Establish Alternate Language Toolkits
- Endorse Other Toolkits

CA7893.139 LJC

THE APPLICATION STYLE GUIDE



- The UNIX® System Standard Look And Feel Is Designed To Make The Power Of The UNIX System Easy and Efficient To Use For Both Non-Technical And Technical Users.
- The AT&T Application Style Guide Is Designed To Make Developing Conforming Programs Easy And Efficient.
- Every Application Developed Using The Application Style Guide Should Be Consistent With Other OPEN LOOK™ Applications So The End User Need Not Relearn Common Actions.
- The Application Style Guide Provides Guidelines On How To Design Applications That Are Easy To Learn, Efficient To Use And Consistent With Other OPEN LOOK Applications

CA5761.110

☐ CAG

OPEN LOOK ™ GRAPHICAL USER INTERFACE

SOURCE PRODUCTS

- OPEN LOOK User Interface
  - X Window System
  - AT&T Window Manager
  - X Server

  Documentation
  - OPEN LOOK User Interface Style Guide
  - OPEN LOOK User Interface Specification

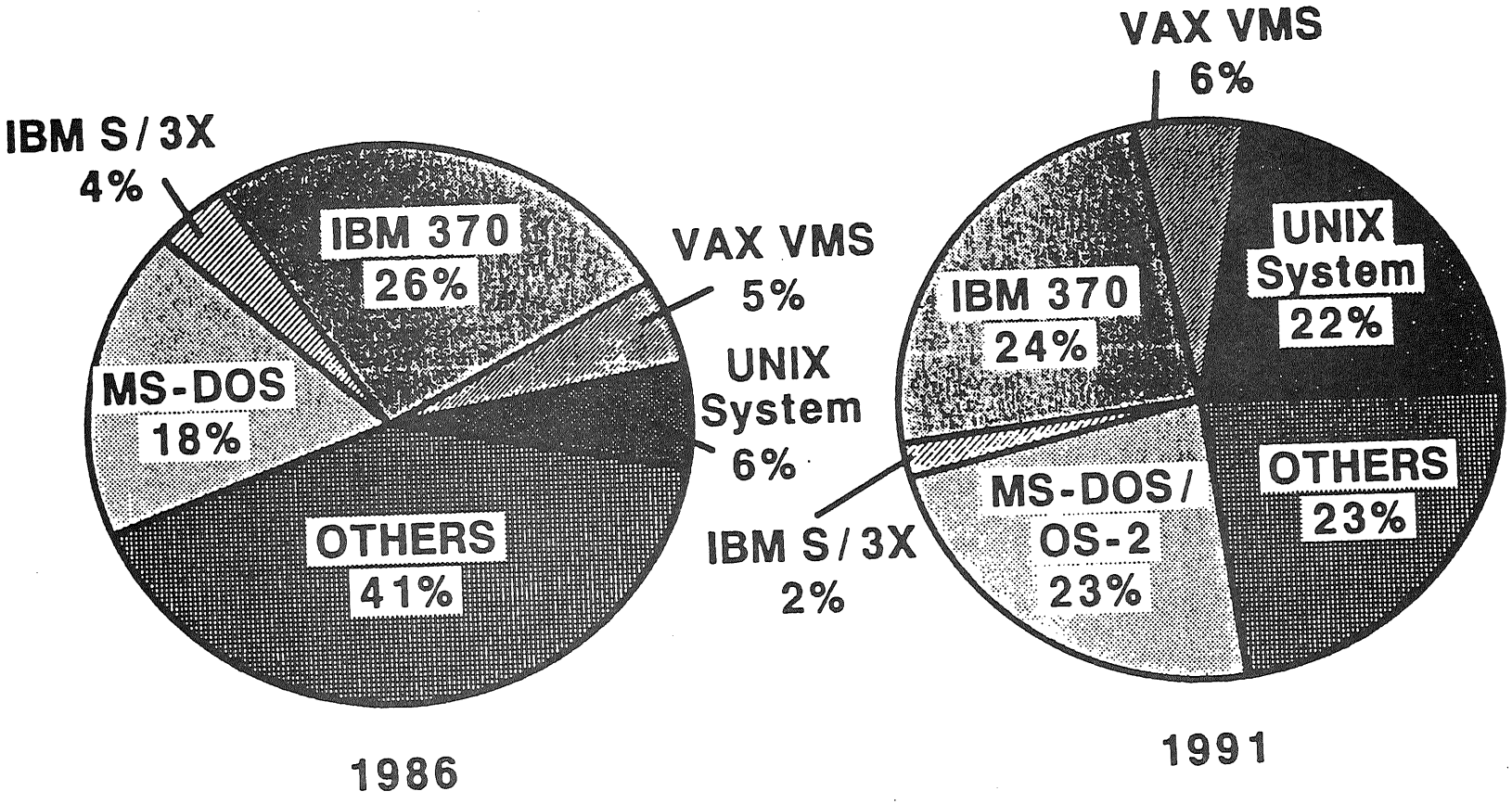
- OPEN LOOK User Interface X Toolkit
  - Toolkit

  Documentation
  - Programming Interface Manual

- OPEN LOOK User Interface NDE Toolkit
  - Toolkit

  Documentation
  - Programming Interface Manual

RAPID GROWTH IN MARKET SHARE OF OPEN SYSTEMS



SOURCE: IDC

OSF and ABI:

Technology and Future

Impact on UNIX

Ross Bott
Pyramid Technology Corporation
September 14, 1988

Applications Development Under UNIX(tm)

Informix:
Currently maintaining 110 different UNIX ports for its products

Oracle:
Currently has 40 UNIX ports
Porting group of 50+

RTI:
Currently has 40+ UNIX ports

SPSS:
17 UNIX ports x 2,000,000 lines of code

Uniplex:
87 Ports

ABI and OSF: Issues

(1) Impact of the proliferation of UNIX versions on software developers and users
  * Does the introduction of OSF just make a bad situation worse?
  * Will the Application Binary Interface solve this proliferation problem?

(2) Binary compatibility (ABI) vs source compatibility (OSF)

(3) What is so difficult about porting an application to a new version of UNIX?

(4) Impact of compatibility and portability on the future of UNIX

PYRAMID TECHNOLOGY

Outline of Presentation

I. Competing pressures brought by the Open Platform philosophy

II. ABI: Current perceptions and an historical perspective

III. ABI: A formal definition

IV. ASI: Application Source Interface

V. ABI vs ASI

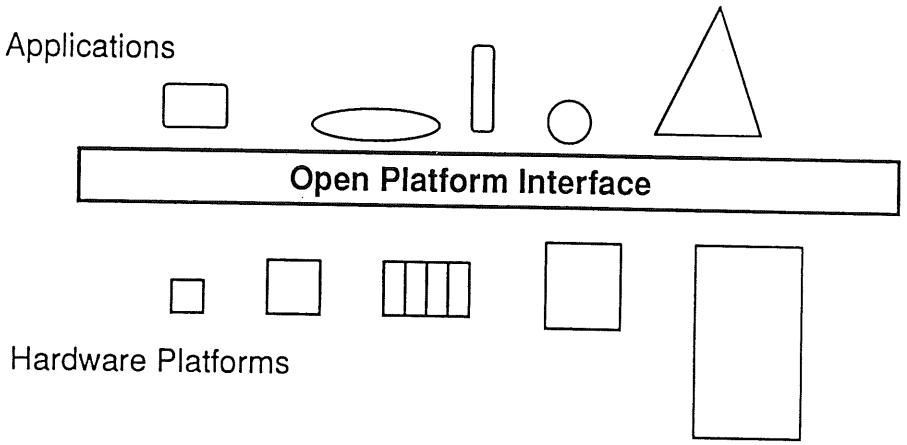
VI. Application porting process

VII. OSF
  Compatibility Levels
  Optional Implementation of Features
  Evolution of Commercial UNIX

VIII. Summary

PYRAMID TECHNOLOGY

Perspective: Open Platform User



PYRAMID TECHNOLOGY

Perspective: Open Platform User

Open Platform Provides:

(1) Wide availability of applications

(2) Continually competitive price-performance

(3) Leveraged system development

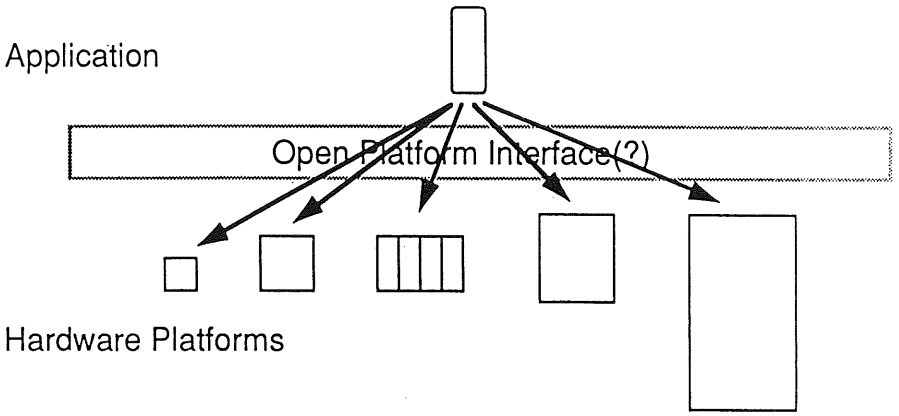
(4) Leveraged application development

(5) Maximum people portability

(6) Not tied to a single hardware manufacturer

PYRAMID TECHNOLOGY

Perspective: Open Platform Software Developer



Perspective: Open Platform Software Developer

Issues:

(1) Huge number of platforms to support

(2) For each:
- (a) new hardware platform
- (b) new release of operating system
- (c) new release of software

the following must be done:
- (a) re-port
- (b) re-regression test
- (c) re-release process
- (d) additional variation to support

(3) Therefore, *the fewer the platforms the better!*

Release Proliferation

Ingres RDBMS

2 releases a year on average

40 UNIX hardware platforms

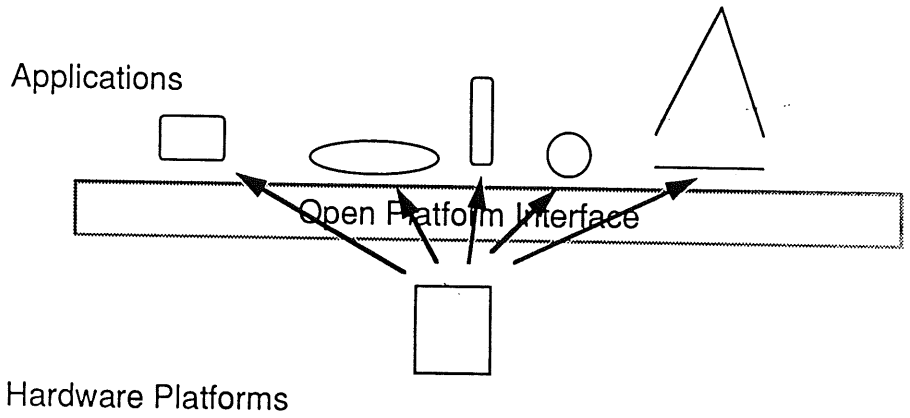
1.5 operating system releases per year for each platform

==> 120 releases each year, even without taking on new platforms!

==> One release every 2 working days

PYRAMID TECHNOLOGY

Perspective: Open Platform Computer Manufacturer



PYRAMID TECHNOLOGY

Perspective: Open Platform Computer Manufacturer

Issues:

(1) Each major software application requires:
- (a) possible porting funding
- (b) possible OEM volume commitments
- (c) machines and support for port
- (d) additional first line software support for system
- (e) continual tracking to assure that latest release is on system

(2) Fewer applications decreases commitments and support requirements.

(3) More applications increases potential customer base.

PYRAMID TECHNOLOGY

Release Proliferation for Manufacturer

Pyramid Technology

335+ *active* software applications

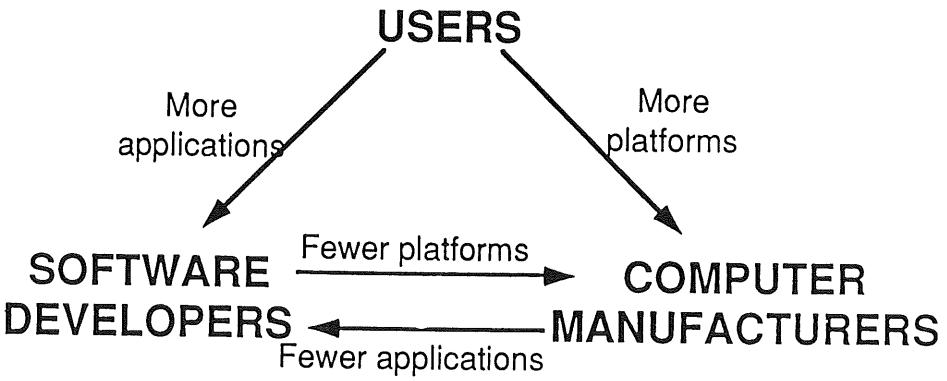
new release every 1-2 years per application

new operating system release every 8 months

==> 335-770 new releases each year, even before adding new applications software

PYRAMID TECHNOLOGY

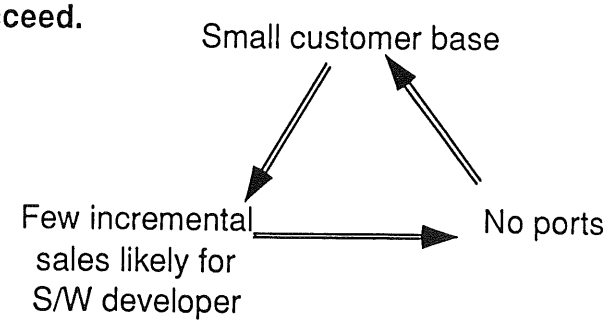
Conflicting Forces Between Perspectives



PYRAMID TECHNOLOGY

Conflicting Forces: Implications

(1) Despite the intent of open platforms, it is becoming increasingly difficult for new computer architectures to succeed.



(2) Problem is magnified by VLSI.

(3) New "deep" software applications are difficult to justify.

PYRAMID TECHNOLOGY

Applications Binary Interface (ABI)

Definition: A specification of a computing environment such that any binary written to this specification will run on any computer system satisfying this specification.

Specification must include:

- Instruction set (IS)
- System call interface (SCI)
- Binary data representation (DR)
- User process layout] (PR)
- Procedure call interface]
- Graphics interface] (UI)
- Window interface]

PYRAMID TECHNOLOGY

ABI: The Political Sell

Users:

(1) *Advantage:* "Assurance" of compatibility of binary applications across ABI platforms

(2) *Advantage:* Less investment in retargeting of in-house applications

Software developers:

(1) *Advantage:* Fewer platforms to worry about.

(2) *Advantage:* Single binary

(3) Option for "shrink wrap software"

PYRAMID TECHNOLOGY

©1988 Pyramid Technology Corporation

ABI: The Political Debate

Computer manufacturers:

(1) *Advantage:* If CPU is an ABI, then one can take advantage of all software ported to that ABI

(2) *Disadvantage:* ABI definition was being defined by a small subset of computer manufacturers. Problems of control and early release of ABI software.

(3) *Disdvantage:* If CPU is not an ABI, then manufacturer becomes an outsider to users and software developers.

(4) *Disadvantage:* Discourages new computer architecture development

(5) *Disadvantage:* Encourages concept of hardware as a commodity.

PYRAMID TECHNOLOGY

PC / MS-DOS as an ABI

How did MS-DOS / PC become an ABI?

(1) De facto standard set by IBM PC

(2) Market large enough and hardware simple enough to support competitive commodity manufacturing.

(3) Selling price of hardware and customer needs require inexpensive software and volume distribution channels. Shrink wrap software becomes a necessity.

(4) Non-expert customer base and shrink wrap software require binary compatibility without complications.

PYRAMID TECHNOLOGY

PC / MS-DOS as an ABI (cont.)

Why does it succeed as an ABI?

(1) Everyone uses the same chip (easy answer)

(2) Standard is not CPU chip but whole system architecture

(3) Performance not critical on most applications ==> little incentive to stray from full standard to boost performance

(4) Third party hardware plug-in requirements

-- Sometimes it really doesn't.

PC / MS-DOS as an ABI (cont.)

Keys

(1) Application simplicity

(2) No requirement for performance tuning

(3) ABI as a full system

(4) Commodity volumes

370/VM as an ABI

Why does it succeed?

(1) De facto standard set by IBM.

(2) Operating system written in a low level language; forces full system compatibility

(3) Huge body of applications, most non-portable.

(4) PCM at the same performance is acceptable: IBM margin is large enough to undercut.

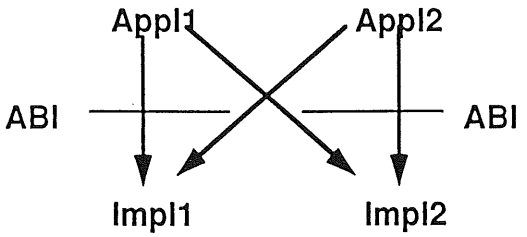
PYRAMID TECHNOLOGY

What Does ABI UNIX Really Buy You?

PYRAMID TECHNOLOGY

ABI: Towards a Formal Definition

Need to formalize:



ABI as an Interface Definition

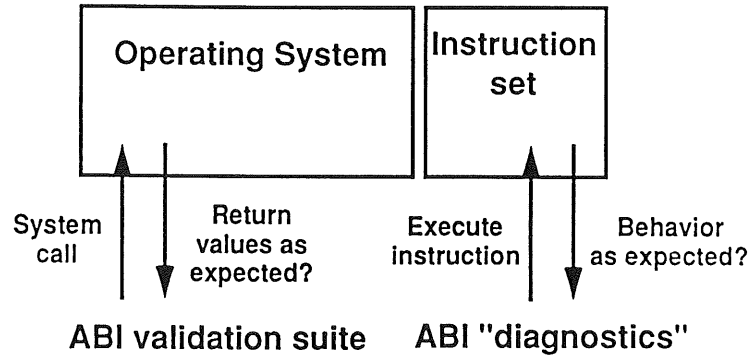
ABI = <IS, SCI, DR, PR, UI>, where

IS = instruction set
SCI = system call interface
DR = binary representation of data
PR = procedure segment setup +
procedure call protocol
UI = window interface + graphic interface

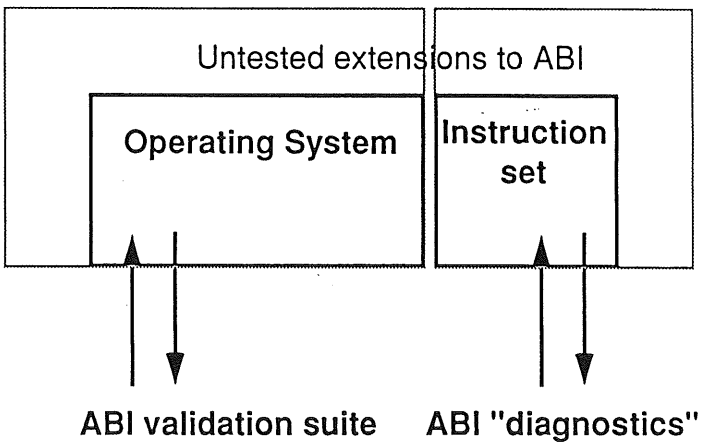
<IS, DR, PR> define a *user computer architecture*

SCI and UI are (virtually) independent of the computer architecture

Testing ABI Compatibility: System Side

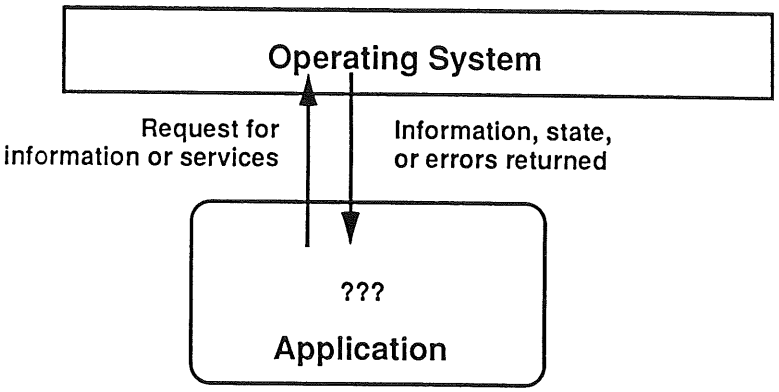


ABI Definition and Validation Suite as a Partial Specification Only



ABI Application Validation

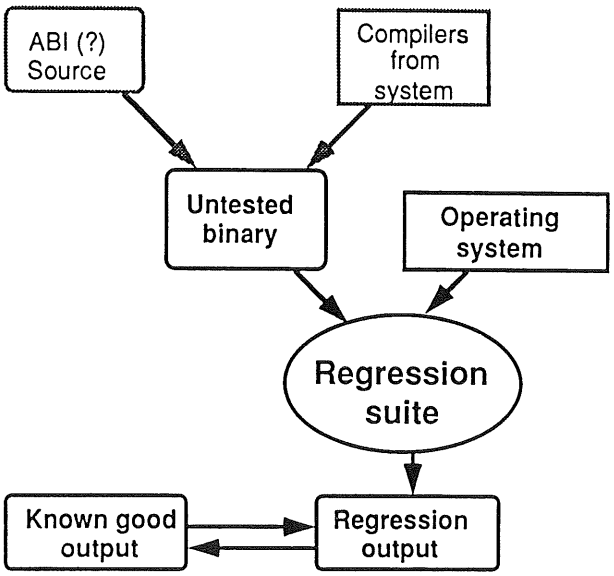
(1) *No single validation suite can exist.*



PYRAMID TECHNOLOGY

ABI Application Validation (cont.)

(2) Indirect validation only is done by regression suite specific to that application.



PYRAMID TECHNOLOGY

ABI As an Operator

Define **abi(Appl)** as

(a) a conversion of the source of application **Appl** to obey **ABI**

(b) a compilation **CG** of the source of **Appl**, where **CG** conforms to **DR** and **PR** and generates code from **IS**.

Thus, **abi(Appl)** is a runnable object on any computer conforming to this **ABI**.

PYRAMID TECHNOLOGY

Application-specific ABI Compatibility

Define: **REGR**-appl as the output from a suite of regression tests for **appl**.

An application is **ABI compatible** iff **REGR**-appl is the same for all known computers satisfying that ABI.

Definition: Two computers **Impl1** and **Impl2** are **ABI-REGR-compatible** with respect to **Appl** iff

REGR[abi(Appl), Impl1] = REGR[abi(Appl), Impl2]

REGR[CG(ABI(Appl),ABI), Impl1] = REGR[CG(ABI(Appl),ABI), Impl2]

REGR[CG(<SCI,UI>(Appl),ABI), Impl1] = REGR[CG(<SCI,UI>(Appl),ABI), Impl2]

PYRAMID TECHNOLOGY

Application Source Interface (ASI)

ASI = <SCI, UI, PS>, where

SCI = system call interface
UI = window interface + graphic interface
PS = porting specification

PS is defined such that no part of the applicaton source is architecture dependent, for an acceptably large class of computer architectures

PYRAMID TECHNOLOGY

Examples of ASI

(1) SVID + X-Windows + GKS + *PS*

(2) X-OPEN (with planned additions) + *PS*

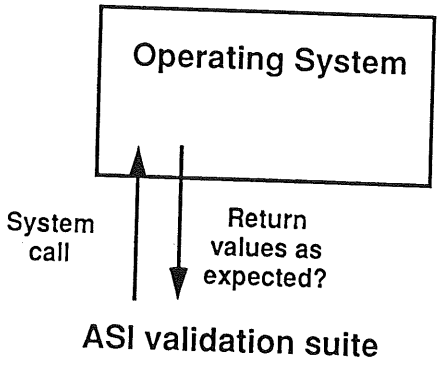
(3) POSIX (with planned additions) + *PS*

(4) OSF Level 1 (1991-2) + *PS*

Keys are
(a) whether SCI is sufficiently powerful
(b) a well articulated PS

PYRAMID TECHNOLOGY

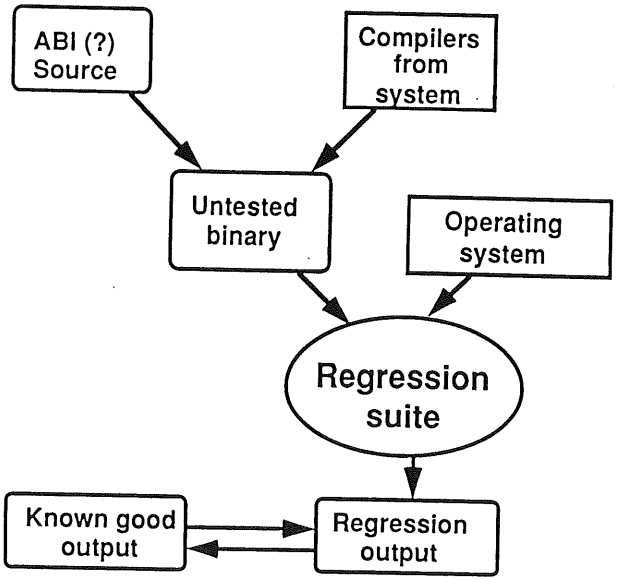
ASI System Validation



PYRAMID TECHNOLOGY

Validation of ASI Application

Indirect validation by regression suite specific to that application.



PYRAMID TECHNOLOGY

Application-specific ASI Compatibility

Definition: Two computers **ASI1** and **ASI2**are
ASI-REGR-compatible with respect to **Appl** iff:

REGR[CG(ASI(Appl), ASI1), ASI1] = REGR[CG(ASI(Appl), ASI2), ASI2]

REGR[CG(<SCI,UI,PS>(Appl),ASI1), ASI1] = REGR[CG(<SCI,UI,PS>(Appl), ASI2), ASI2]

PYRAMID TECHNOLOGY

ABI vs ASI

REGR[CG(<SCI,UI>(Appl),ABI), Impl1] = REGR[CG(<SCI,UI>(Appl),ABI), Impl2]

REGR[CG(<SCI,UI,PS>(Appl),ASI1), ASI1] = REGR[CG(<SCI,UI,PS>(Appl), ASI2), ASI2]

Identical:

(1) Application does not change.
(2) Standards that application must adhere to are architecture independent.
(3) Regression test is (in theory) a formality.

Differences:

(1) With ABI, CG targets to same instruction set, and must be done only once. With ASI, CG must be done once for each side of the equation.
(2) ABI doesn't need PS.

PYRAMID TECHNOLOGY

ABI vs ASI: In Theory

As interface definition:

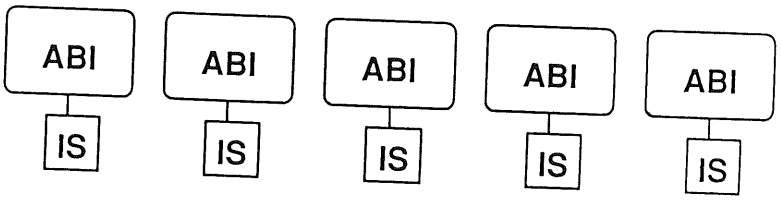
ABI = ASI + instruction set specification

As a product release process:

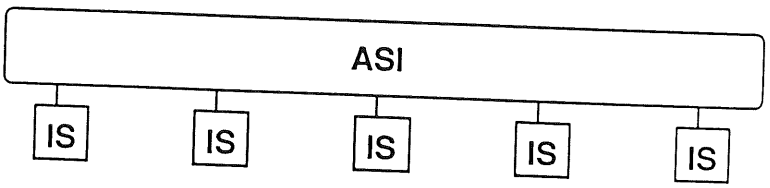
ABI = ASI - incremental compile

PYRAMID TECHNOLOGY

Resolving the Differences: Multiple ABI's?



is equivalent to



for small numbers of machines from each manufacturer.
Thus, non-merchant chip ABI's are nearly meaningless.

PYRAMID TECHNOLOGY

ABI vs ASI:
How Does Theory Differ From Perception?

Software developer: A single compile is a very big deal. No manufacturer agrees upon compiler, OS, or debugger standards. Separate releases for each ASI are a big headache.

User: For internal applications, no perceived need to compile. For external applications, nothing.

Computer manufacturer: A chance of running binaries from untested third party developers.

PYRAMID TECHNOLOGY

Myth No. 1

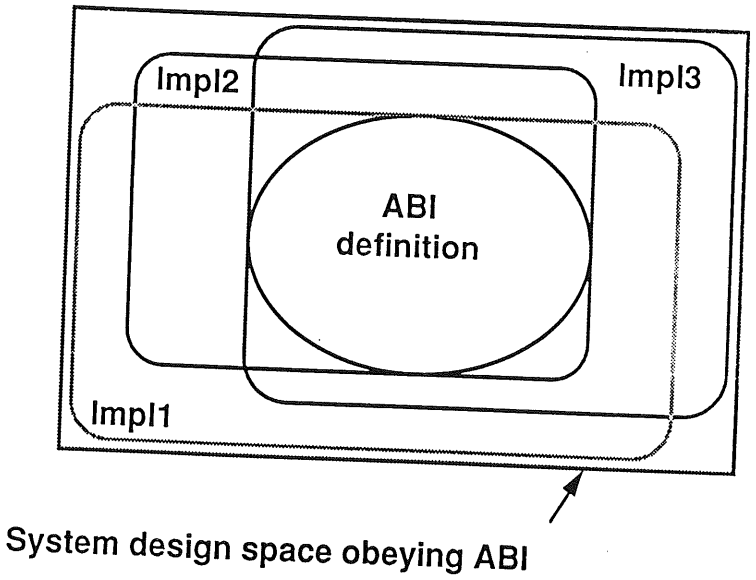
ABI guarantees that a binary application will run on any ABI-compatible computer.

False: Application can use feature outside of ABI.

Example: On VAX, for p=0, *p=0.

PYRAMID TECHNOLOGY

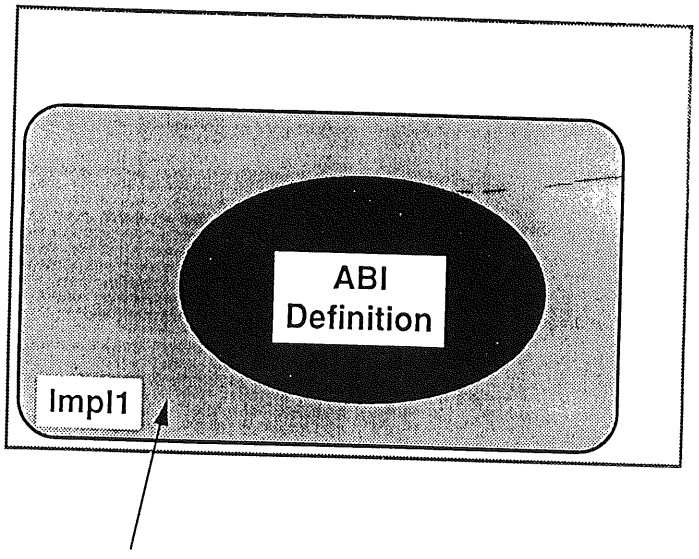
ABI as a Partial Constraint Definition



System design space obeying ABI

PYRAMID
TECHNOLOGY

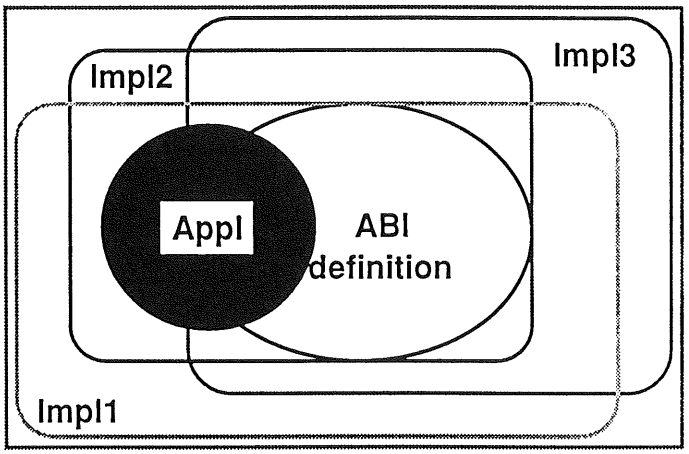
From ABI Definition to ABI System



Definition incompleteness
Extension to full system
Feature enhancements
Performance enhancements

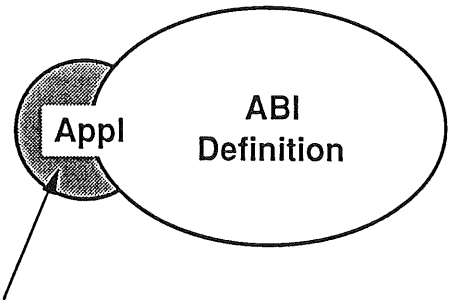
PYRAMID
TECHNOLOGY

ABI Application
Violating Binary Compatibility



PYRAMID
TECHNOLOGY

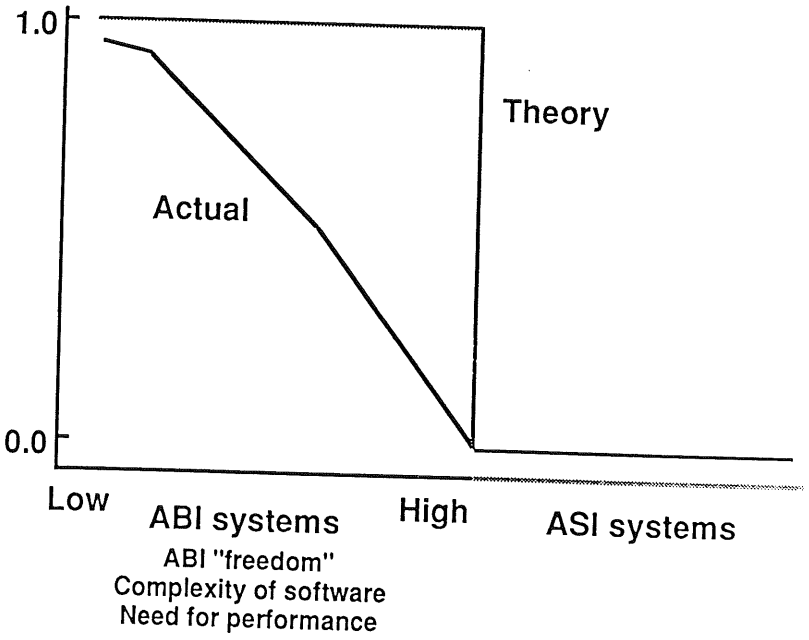
Why An Application Can
Stray Beyond ABI Definition



Need for additional features
(application complexity)
Performance requirements
Unintentional straying
Cannot test for application ABI compatibility

PYRAMID
TECHNOLOGY

Cross-System Binary Compatibility



PYRAMID TECHNOLOGY

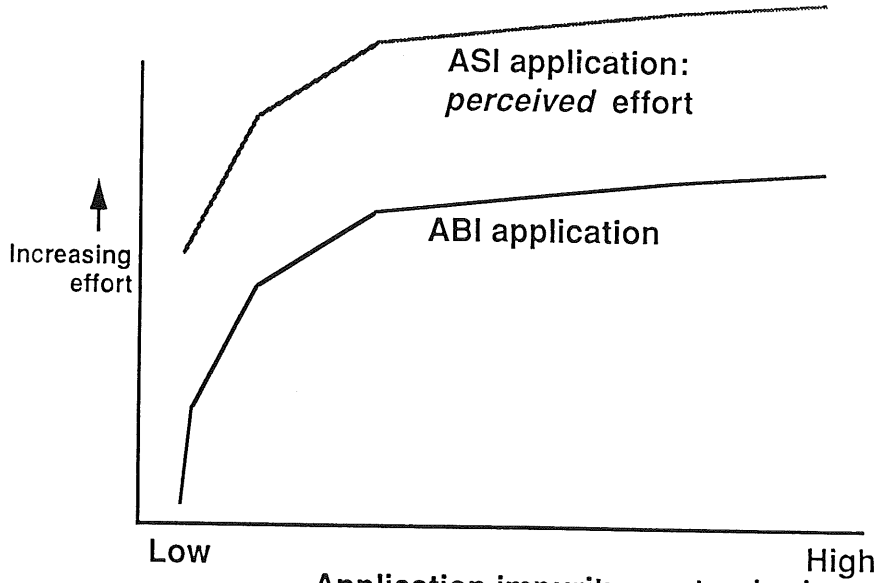
Myth No. 2

An ASI would imply a greater porting effort than and ABI.

False: If application obeys porting specification and computer system provides standard front end to compiler, then "port" for ASI is a recompile.

PYRAMID TECHNOLOGY

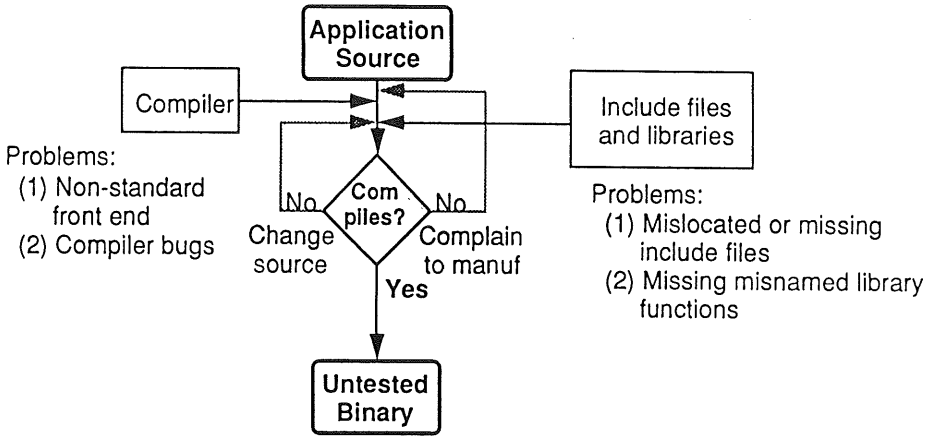
Perceived Porting Effort Required of Software Developer



Application impurity vs standard
System impurity vs standard
Application complexity
Need for performance

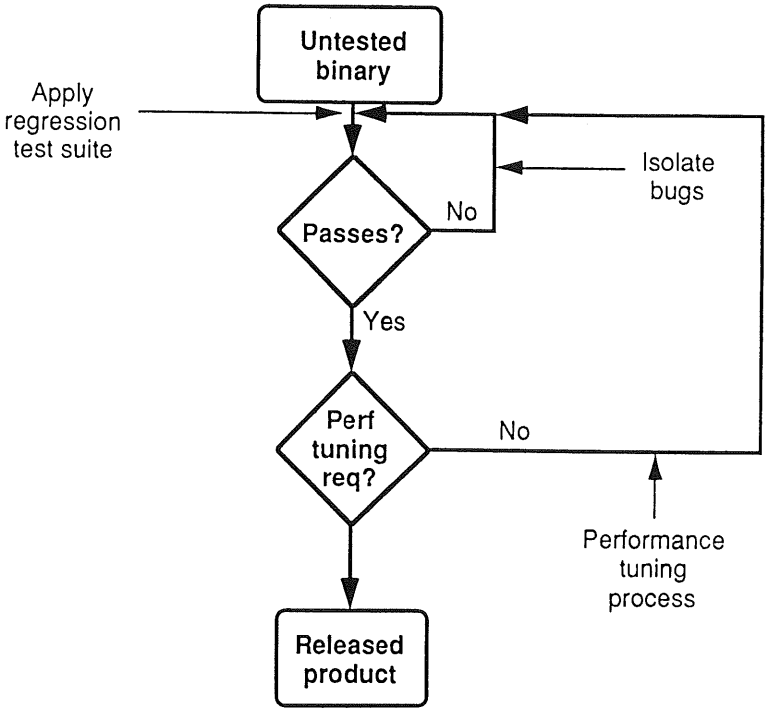
PYRAMID TECHNOLOGY

Application Porting Process

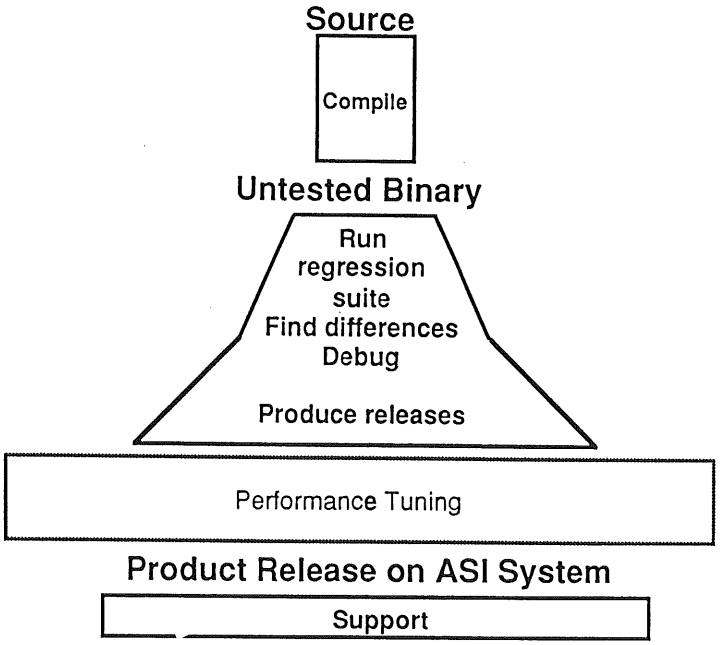


PYRAMID TECHNOLOGY

Application Porting Process (cont.)



Porting Effort to Different ASI (C Language)



Porting Process: Example

Ingres RDBMS:

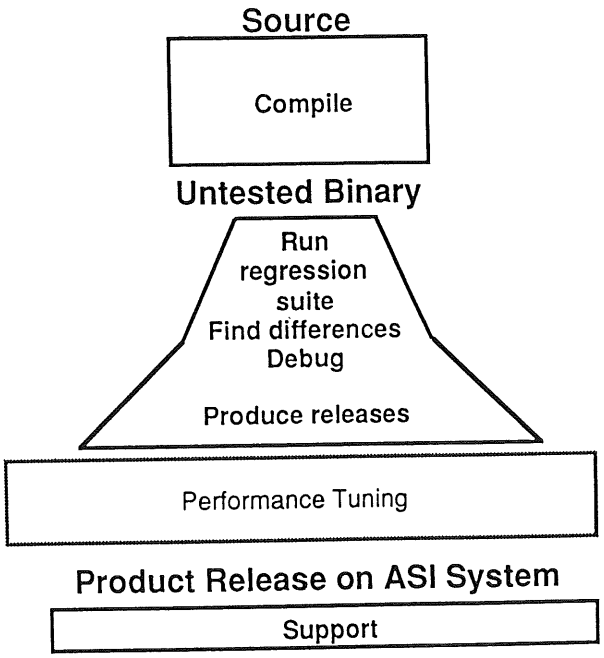
Stage 1 (new port): 5 days to 3 months
(re-port): 2-3 days

Stage 2 (regression): 2-3 days CPU time
7-8 days diff & debug

Stage 3 (tuning): 2-12 months

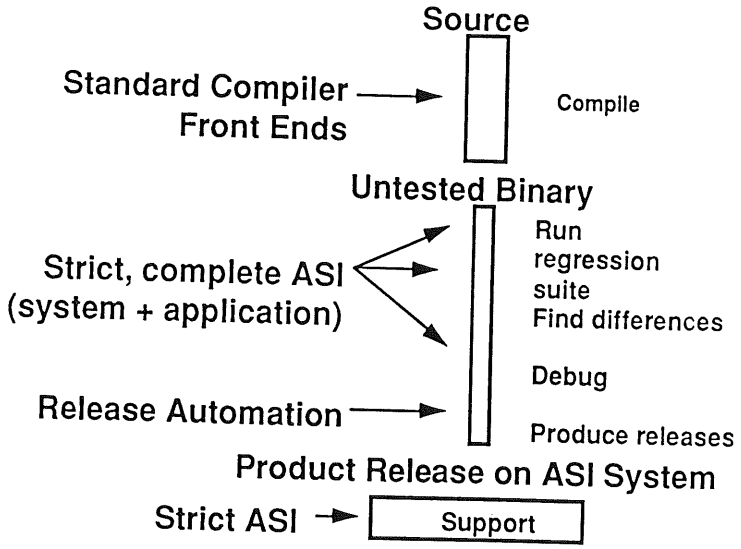
PYRAMID TECHNOLOGY

Porting Effort to Different ASI (Fortran Language)



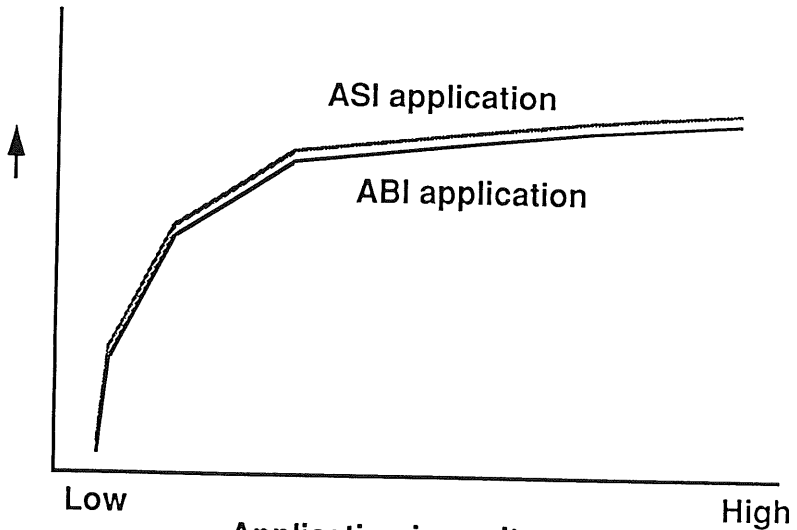
PYRAMID TECHNOLOGY

Towards an Ideal ASI Porting Process



PYRAMID TECHNOLOGY

Porting Effort Required With Strong ASI and Porting System (PS)



Application impurity vs standard
System impurity vs standard
Application complexity
Need for performance

PYRAMID TECHNOLOGY

Myth No. 3

ABI allows UNIX to move into the Age of "Shrink-Wrap Software"

Partially false:

Shrink wrap software =
(1) binary compatibility
(2) cost
(3) low complexity & performance not an issue

PYRAMID TECHNOLOGY

Commodity-based ABI's

	PC / MS-DOS	Low end ABI Unix
Shrink wrap software?	Yes	Soon
Application complexity	Low to moderate	Low to moderate
Sys Stand exists?	Yes	Some cases
useful?	Yes	Yes
Commodity hardware?	Yes	Coming soon
High gross margins?	No	No
Performance important?	Sometimes	Sometimes

PYRAMID TECHNOLOGY

Price Leverage ABI

	High end ABI Unix	370 VM / MVS
Shrink wrap software?	No	No
Application complexity	Moderate to high	Moderate to high
Sys Stand exists?	No	No
useful?	No(?)	Yes
Commodity hardware?	No	No
High gross margins?	No	Yes
Performance important?	Yes	Sometimes

PYRAMID TECHNOLOGY

Open Software Foundation: The Political Reasons For Its Creation

* Counter to the power move of AT&T and Sun

* ABI to be defined by a small subset of developers

* (Successful) attempt to reopen and reunify UNIX

PYRAMID TECHNOLOGY

Technical Goals for OSF UNIX

(1) Source distributed; available to many hardware platforms (OSF as ASI)

(2) Adherence to standards
* verification suites are crucial

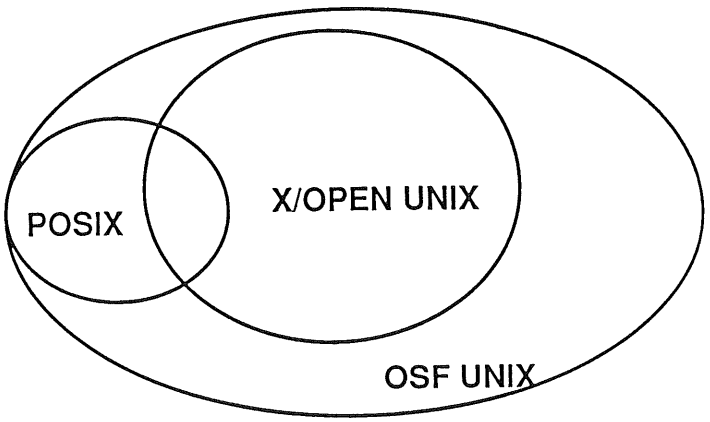
(3) Freedom to choose what to implement

(4) Significant feature enhancements to UNIX

PYRAMID TECHNOLOGY

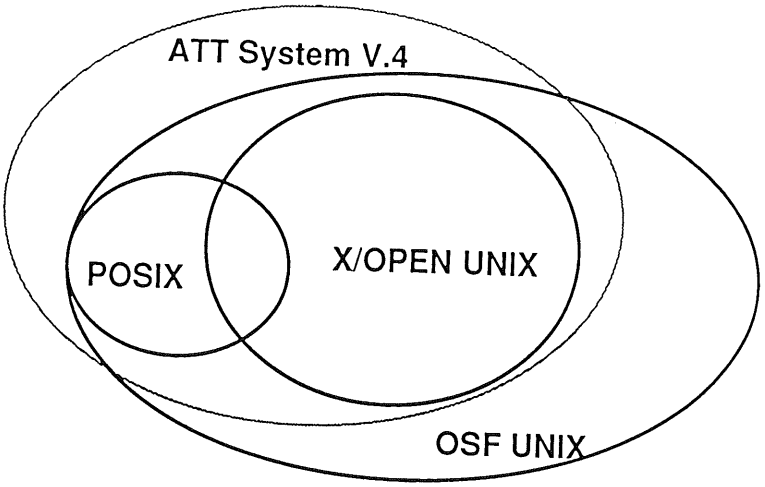
Adherence to Standards

OSF as a superset of POSIX and X/OPEN UNIX's



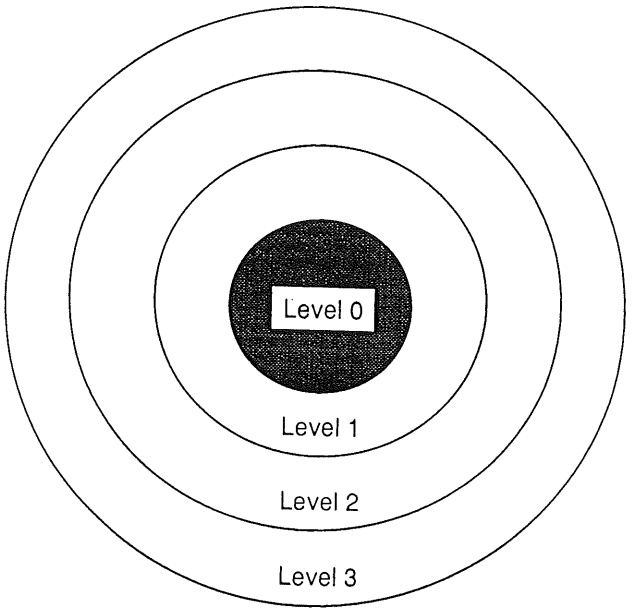
PYRAMID TECHNOLOGY

"Danger" of Multiple UNIX's



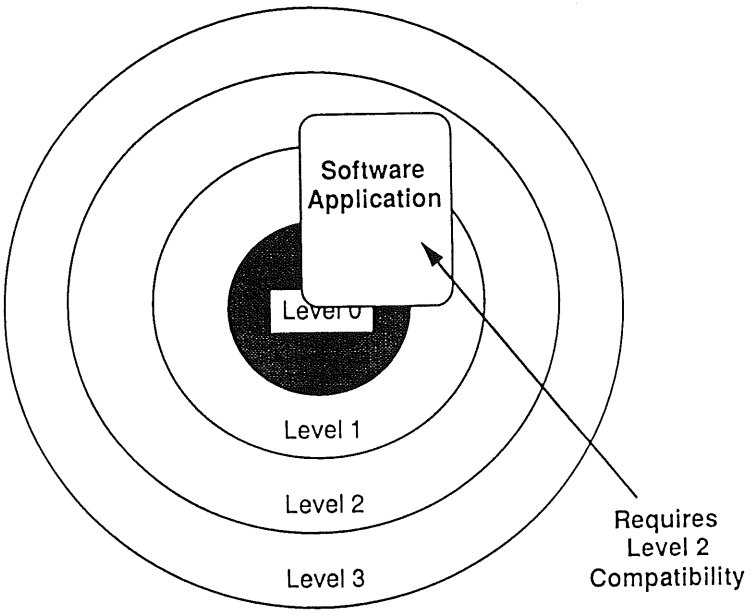
PYRAMID TECHNOLOGY

UNIX Compatibility Levels



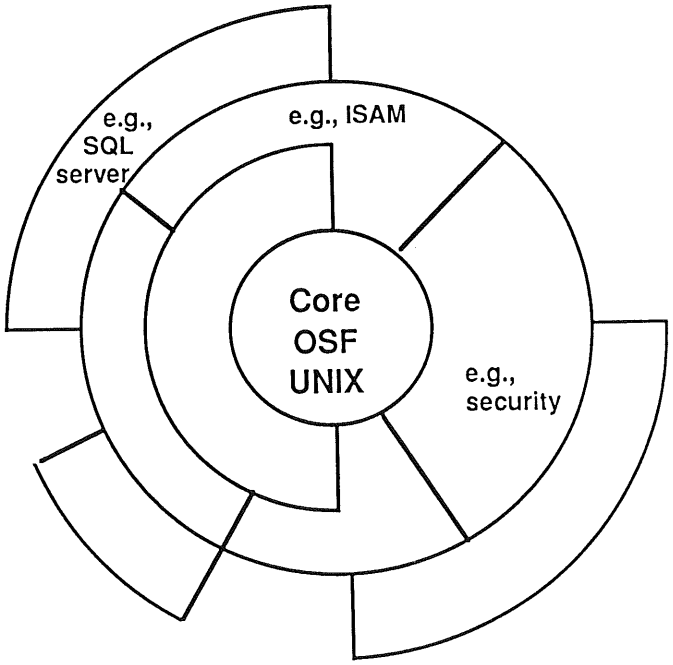
PYRAMID TECHNOLOGY

Compatibility Level of a Software Application



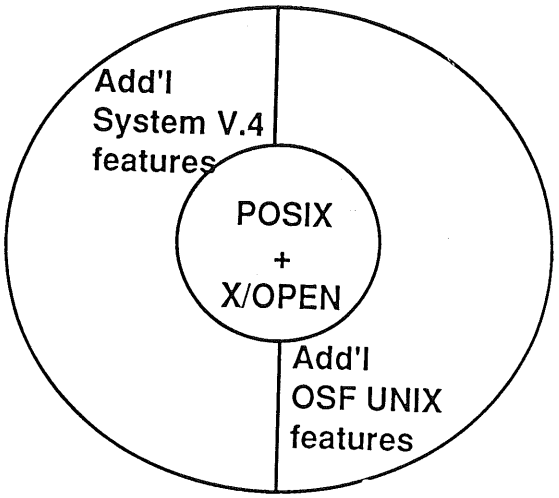
PYRAMID
TECHNOLOGY

"Freedom" of Implementation of Optional Features



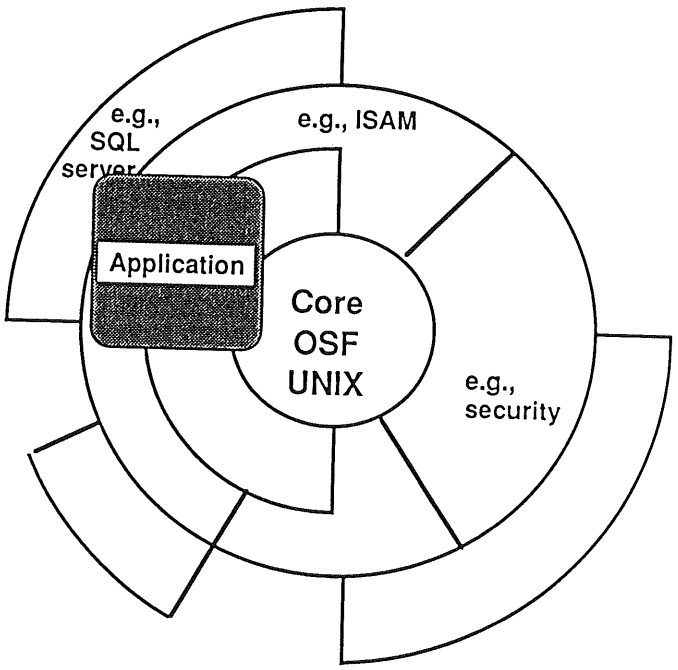
PYRAMID
TECHNOLOGY

Two UNIX's:
An Example of Optional Freedom



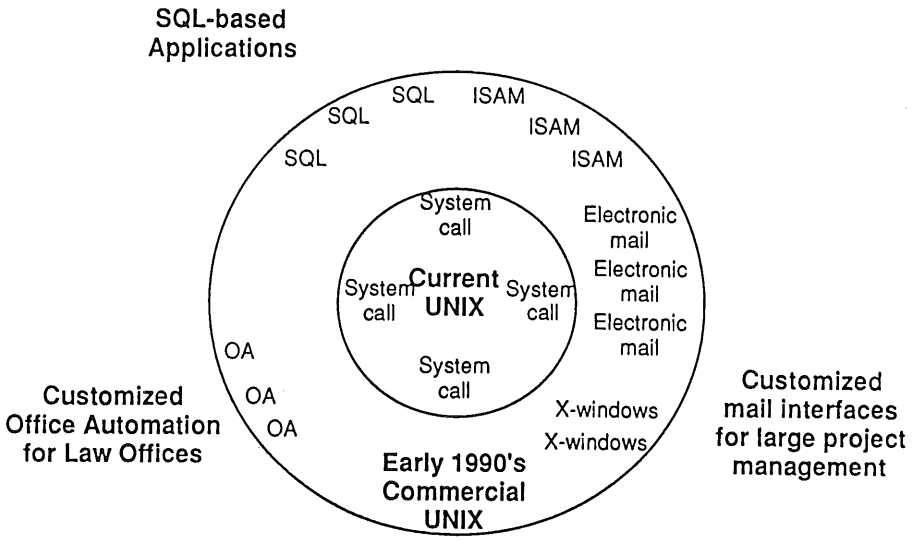
PYRAMID
TECHNOLOGY

Effect of Option Freedom on Applications



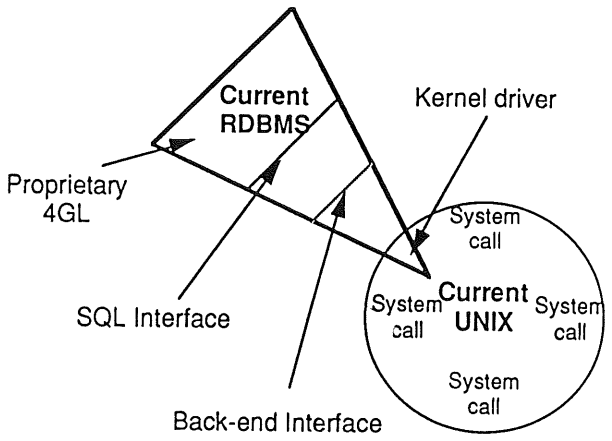
PYRAMID
TECHNOLOGY

Evolution of Commercial UNIX



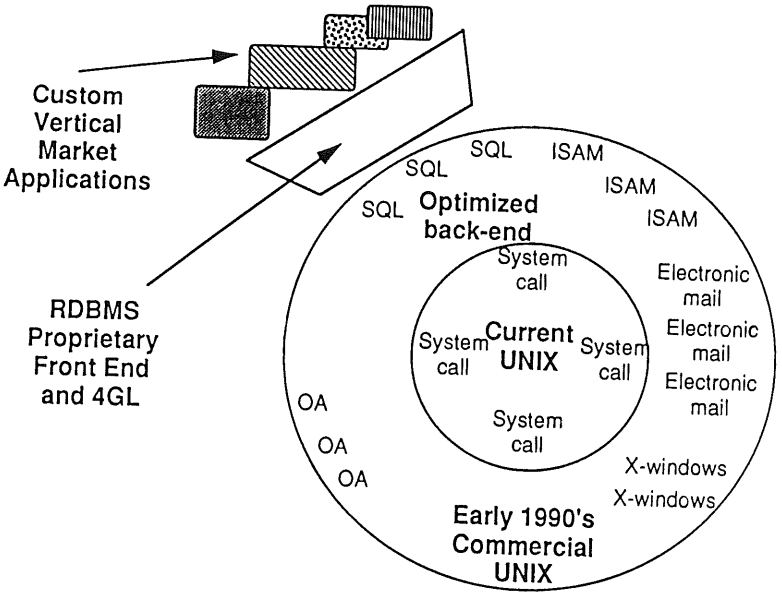
PYRAMID TECHNOLOGY

UNIX and RDBMSs: 1988 Relationship



PYRAMID TECHNOLOGY

UNIX and RDBMSs: 1990s Relationship



PYRAMID
TECHNOLOGY

UNIX and RDBMSs: 1990s Relationship

Back-end contained within the UNIX operating system

High speed transaction performance built in to OS

PYRAMID
TECHNOLOGY

Summary

(1) Absolute ABI is an illusion.

(2) ABI UNIX is possible at the low end (commodity ABI)

(3) ABI UNIX unlikely at the high end
* performance tuning is critical
* no price margins to work with
* a disservice to creativity in computer architectures

(4) A well specified and conformed to ASI approaches an ABI in effective utility

(5) OSF optional implementation approach dangerous unless extremely well specified.

(6) ABI pushes trend towards hardware as commodity.

(7) OSF pushes trend toward software as a commodity.

PYRAMID TECHNOLOGY

Research and Entreprenurial Opportunities

(1) Regression testing process as:

(a) an expert system

(b) a completely automated process

(2) Computer assisted release management

(3) A theory of porting (given a space of computer architectures)

(4) High efficiency object to object translation

PYRAMID TECHNOLOGY

;login:

The USENIX Association Newsletter

Volume 13, Number 4 **July/August 1988**

CONTENTS

The closing date for submissions for the next issue of *;login:* is August 26, 1988

THE PROFESSIONAL AND TECHNICAL
UNIX® ASSOCIATION

Charge Number Accounting Without Kernel Modifications

Paul E. McKenney

SRI International
Computer Systems & Services

ABSTRACT

The Golden Age of timesharing may be over, but there still are some hosts that must perform resource usage accounting.

There have been many charge number accounting schemes implemented on UNIX, however, they typically require modifications to the UNIX kernel and to system programs such as *login*.

Sri-unix had been running such a system for a number of years, but it was becoming very expensive to maintain, partly due to the age of the code and the number of programmers who had modified it, but also because the modifications had to be reinstalled into each new release of the operating system. Therefore, we at SRI decided to rewrite the accounting system so that it would run on a vanilla BSD system (without modifications to the kernel or to systems programs). This project was successful; the resulting accounting system has been running on sri-unix for over a year.

This paper will give an overview of the implementation, features, and limitations of this system, and will discuss some features of UNIX that helped (and hindered!) its development.

1. Introduction

Sri-unix is a "central" machine that is owned by SRI as a whole. Its continued existence is justified only by the willingness of its users to pay for the privilege of using it. In this day of personal computers and workstations, it might be helpful to list some reasons why people would want to use a centrally located super-mini:

- Low up-front cost.
- Availability of user consultants.
- Disk backups and other administrative chores performed by central staff.
- Good network connectivity that might be expensive to reproduce elsewhere.

Most of sri-unix's users fall into these categories:

- *Mail and network news readers.* These users would probably be satisfied with a normal accounting system, as they typically use a single charge number for all of their work.
- *Programmers.* Again, these users would probably be satisfied with a normal accounting system.
- *Technical writers.* These users need charge number accounting, since they typically prepare reports and proposals for many ongoing projects. Furthermore, these users wanted to be able to change charge numbers in mid-session, as logging off of and onto a busy system can be time-consuming.

While there are relatively few technical writers on the system, the technical writers make much heavier use of the system than most other users do. They thus make a much heavier contribution to the system's revenues than do the other users. Suffice it to say that we were more than willing to implement an accounting system to make life easier for them.

2. The Old Accounting System

The old accounting system did charge number accounting that was based on the standard System V accounting system [2]. It billed for connect time, CPU time, disk space, and printer usage. Connect time and CPU time were billed at a lower rate during non-prime time and on holidays, and could be billed to different charge numbers (at the user's discretion) on a session-by-session basis. Disk space and printer usage were billed at a flat rate to the user's default charge number.

The old system had the following drawbacks:

- It had been modified over the years by a legion of programmers. The code was very difficult to read and maintain, and had accumulated an annoying number of idiosyncrasies and bugs.
- It required users to log out and back in to change their charge number. This mis-feature did not make the technical writers happy.
- It relied on a specially modified version of *login* to collect and record the charge number information. This modification had to be reapplied every time the operating system was upgraded, which diverted scarce systems programming resources from other projects and scarce budgetary resources to source license renewals.
- There was no invoicing to the users, thus they were unsure of how much money they were spending until their charges worked their way through SRI's main MIS system some weeks later.

3. Requirements for the New Accounting System

The new accounting system was required to:

- Run on a vanilla BSD 4.2 version of UNIX. No modifications to the kernel or to systems programs were allowed.
- Charge for connect time, CPU time, disk space, and printer usage.
- Bill CPU and connect time at a lower rate during non-prime time and on holidays.
- Allow the user to specify a charge number at any time during a login session, and have all subsequent connect and CPU time used during that session billed to that charge number.
- Maintain lists of valid charge numbers. Each valid charge number must have a list of users who are permitted to use it. A separate charge number list must be kept for each UNIX group, and it must be possible to designate a user to be the group administrator for a given group. A group administrator must be able to modify the list of valid charge numbers for his group. The supervisor for a group is usually designated as that group's administrator.
- Produce per-command and per-user usage reports each week.
- Produce a billing tape each week that can be used as input into SRI's MIS system.
- Send weekly invoices to users, at their discretion. A user should also be able to specify that his invoice should be sent to someone else (e.g., his supervisor).
- Allow a user to see how much the current session has cost so far.
- Of course, the system should run with as little human intervention as possible.

4. The User's View of the New Accounting System

4.1. Setting the Charge Number

The *setcharge* program is the normal user's main interface to the accounting system. This program may be run at any time during a login session. It will query the user for a charge number, possibly offering a default. The charge number entered by the user is checked against a list of valid charge numbers for that user. If the charge number is accepted, all subsequent CPU and connect time for the current login session will be billed to that charge number.

If the user never runs the *setcharge* program, the session will be billed to his default charge number. "The accounting system *always* gets its money."

4.2. Weekly Invoices

A user can get weekly invoices automatically mailed to him by placing an empty file named *.InvoiceWeekly* into his home directory. He may also place a list of mail addresses into the *.InvoiceWeekly* file. This will cause weekly invoices to be mailed to each address in the list.

For example, suppose that Mary is Peter's supervisor, and that she wants to see his weekly invoices. If Peter places the following into his *.InvoiceWeekly* file:

peter
mary

they will both get a copy of Peter's weekly invoices.

4.3. Cost of Current Session

A csh script named *SessionCost* (based on the csh *time* command) may be sourced to determine the connect and CPU charges incurred so far for the current session. The following is an example of *SessionCost*'s output:

CPU and connect cost for session excluding running background processes:

$1.02 CPU, $4.83 Connect, $5.85 Total

Since UNIX does not record the resource usage of a process until the process terminates, the CPU time printed by the *SessionCost* command does not include usage by processes still running in the background. Therefore, users should not be running any background jobs when using *SessionCost* to determine the cost of running a program.

4.4. Charge Number Validation

Each UNIX group has its own list of valid charge numbers, and its own set of lists of users who are allowed to bill to each of those charge numbers. The system administrator may designate a user as the administrator for a group.

The group administrator can run the *chacct* program to add and delete charge numbers for the group, and to specify which of the users in that group will be allowed to use each charge number. The group administrator can also designate a default charge number for each user in his group. All of a user's disk and printer usage is billed to his default charge number.

5. The System's View of the New Accounting System

This section will concentrate on how the accounting system associates CPU and connect time usage with the proper charge number.

The *setcharge* program records charge numbers onto the file */usr/adm/chgacct*. The format of each *chgacct* record is as follows:

Position	Field Name	Description
0-7	ut_line	Terminal line name
8-15	ut_name	User name
16-31	ut_host	Charge number
32-35	ut_time	Time of charge number change

The astute reader may have noticed that this is really a *utmp* record with the charge number placed into the *ut_host* field (which normally contains the name of the remote host for remote login sessions). Stealing the *utmp* record format allows all the utilities that currently work with the */etc/wtmp* file to be used on the */usr/adm/chgacct* file.

Each night, records from the *pacct* and *wtmp* files are correlated with the *chgacct* records. *pacct* records are output by the kernel each time a process terminates. They contain the user ID, the time the process was created, and the name of the process's controlling terminal (as well as many other things, including the amount of CPU time that the process used). *wtmp* records are output by *login* (when a user is logged in) and by *init* (when a user logs out). They contain the user name, the terminal name, the name of the remote host (for telnet logins), and the time that the login/logout occurred.

All three files are converted to ASCII so that they may be processed by the system *sort* utility. The *wtmp* records for the beginning and end of a login session are combined into a single record that represents the full login session. They are sorted by time within terminal line within user as shown in the following table:

User Name	Terminal Name	Time
fred	ttyi08	08:43
		14:21
	ttyi1a	07:30
		10:35
		12:54
harry	ttyi03	16:36
	ttyi11	01:12
		05:42

This ordering allows a simple merge program to associate the proper charge number with a given *pacct* or login session record. Of course, it may be necessary to split a login session record and associate each of the pieces with a different charge number. It is not necessary to do this with *pacct* records because each process is billed to the charge number in effect when that process was created.

Note that this method of assigning charge numbers to *pacct* records requires the user be careful when running shell scripts in the background. If the user starts up such a script, then changes the charge number, any processes started by the script after the charge number change will be billed to the new charge number.

6. Help and Hindrance from UNIX

The accounting system benefitted from the concepts of pipes, filters, and shell scripts in much the same way that many other applications have in the past.

The following sections describe some of the problems that UNIX presented.

6.1. Time Base

There is no absolutely trustworthy time base in UNIX, as the time of day can be changed at any time by anyone with super-user privileges. Since much of the billing is based on elapsed times, an accurate time base is very important to the accounting system.

The *date* program does leave evidence of date changes in the */etc/wtmp* file, but it is not always possible to correlate these change records with the timestamps in files created by the accounting system. For example, if the time is set back five minutes at 8:30, then there will be two different *8:27*s. How is the poor application program to tell which of the two possible *8:27*s is meant?

Our work-around for this problem was straightforward – we prohibit setting of the date while in multi-user mode. However, it would be much more convenient for our operators and our users if we could allow the date to be changed while in multi-user mode. This could be done more easily if there was a counter that was faithfully incremented with the passage of time, independent of the the time-of-day clock (perhaps an ITIMER_MONOTONIC?).

Note that an accurate time base is important to any program that needs to make elapsed-time measurements, including data collection programs and real-time systems.

6.2. Single-precision awk

awk keeps its numbers in single-precision floating-point variables. This means that you cannot do arithmetic on `time_t`s in *awk*. (Note that this has been fixed by some vendors; I applaud them.)

We worked around this problem by writing more C programs than originally planned, with attendant schedule slippage.

6.3. Explicit Support for Charge Numbers

Explicit support for charge numbers in the kernel would make this whole exercise trivial (aside from the maintenance of the kernel code). [1] took this approach.

The printer spooling system and the disk quota system also do not know about charge numbers. An accounting system that needed to do charge number accounting for printing and for disk usage could benefit from a change in this situation.

Bibliography

1. Eaton, Charles K.: "Project Accounting on UNICOS," USENIX Conference Proceedings, Winter 1988.
2. Knutsen, Andrew: "SRI Modifications to Pyramid/4.2bsd/SysV Accounting," SRI, August 1985.
3. Source Code for BSD 4.3, Berkeley, May 1986.

Presenting a Single System Image with Fine Granularity Mounts

Charles H. Sauer

IBM Advanced Engineering Systems
Austin, Texas 78758

ABSTRACT

In distributed system environments, a variety of administrative environments (*system images*) can be presented, reflecting different user requirements and administrative objectives. One of the most important system images is the so called "single system image." This paper provides a context and definition for single system image. It describes an effective approach to collecting multiple UNIX systems into a single system image, based on simple use of remote mounts at fine granularities, including individual files. The approach is designed to allow for replication of administrative files, e.g., */etc/passwd*, and graceful reconfiguration of the system to accommodate planned outages and respond to unplanned outages. Experiences with this approach and AIX† Distributed Services are summarized.

Introduction

In a distributed system environment, individual machines usually perform roles as servers (file, print, name, ...) and/or clients. Subsets of machines may be associated into administrative groups or the associations between machines may remain primarily pairwise and *ad hoc*. Figure 1 illustrates a typical software development environment. Some machines provide services to all other machines in the organization, e.g., network news, source control, special devices, etc. Some machines are administered directly by their owners and have only loose associations with other machines, e.g., the organization wide servers. Many of the other machines are collected into *single system images*, based on suborganizations. In the figure, single system images are represented by dashed boxes, intended to suggest "virtual" laboratories, virtual floors of a building, or virtual buildings, as appropriate to the organization. (It is likely that the association of the machines into single system images will be based on organizational functions and boundaries, not physical boundaries.) Within a single system image, machines are administered as a group, with the intent that users can use any of the machines equivalently. There will be inherent exceptions to this, e.g., some machines will have color displays and others will have monochrome displays. And even where the hardware configurations are the same, the end users will usually be able to distinguish one machine from another, e.g., by querying a machine readable serial number. But a successful approach will give users the illusion that all the machines are the same under most circumstances:

user accounts/passwords. A user can login to any of the machines using the same login name and the same password. Regardless of which machine is used, the user has the same home directory and execution environment. When the user changes his/her password, using the standard *passwd* command, the change is effective immediately on all machines in the single system image. When an administrator adds a new account, this is done once for all the machines.

availability. Even though one or more of the machines is unavailable, the rest of the machines are still able to function together and present the same system image, except for resources which exist only on unavailable machines. A machine which cannot connect to other machines is still usable.

† AIX is a trademark of International Business Machines Corporation.

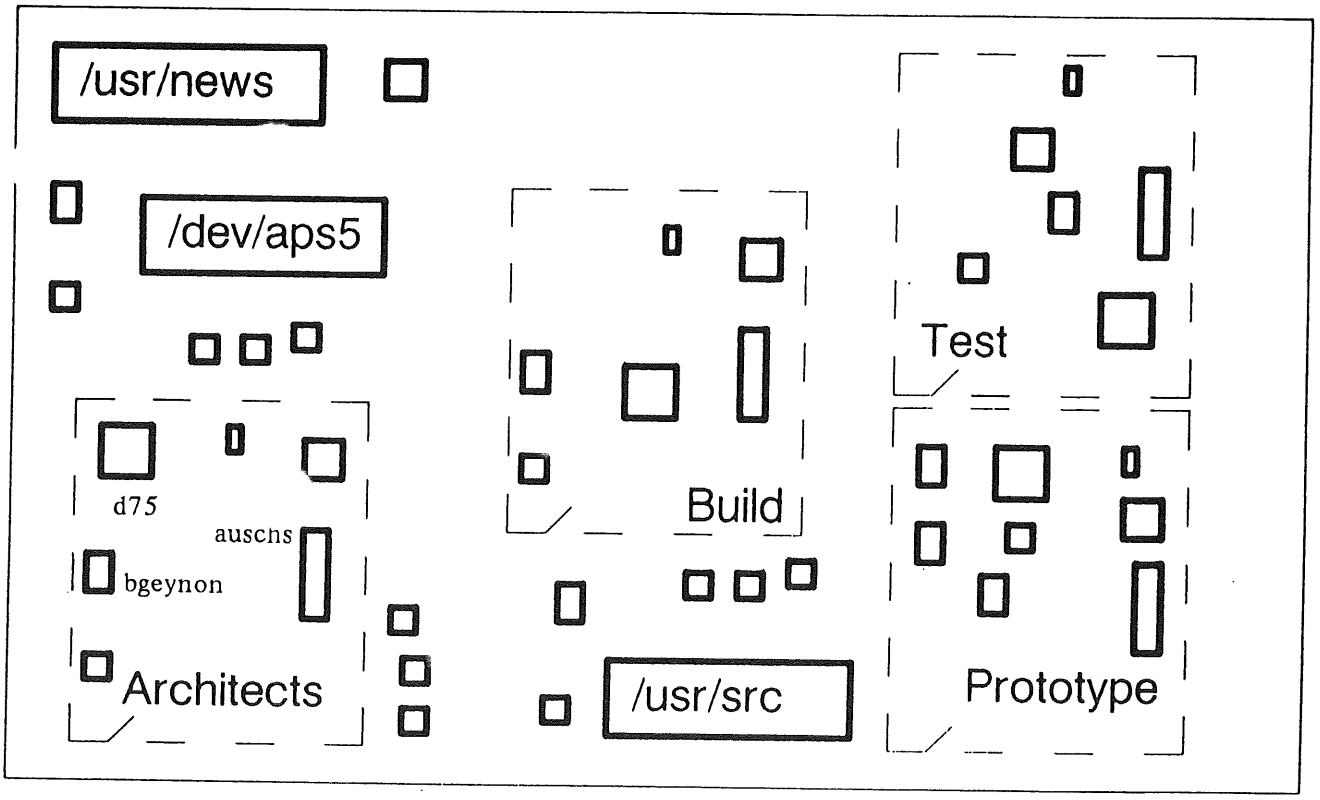


Figure 1: Associations of machines

administered services. System wide functions, e.g., print and mail service, function the same from machine to machine. If mail is sent to a particular user, it can be seen/handled on any of the machines.

This is not an exhaustive list, but is intended to be indicative. Wherever possible, the administrator should view the collection of machines as if it were one machine and use the same procedures that would be used on a single machine. We will use the above characteristics as an operational definition of "single system image" and discuss an approach which we believe is effective in meeting the definition.

Distributed Services (DS) provides distributed operating system capabilities for the AIX operating system. These include distributed file services with local/remote transparency, distributed interprocess communication and a number of administrative services. For background information on DS, see Sauer *et al* [1,2,3] and Levitt [4]. One of the design goals of DS was to provide support for mixed administrative environments, such as the one depicted in Figure 1, using the same protocols and conventions across the administrative environment. One of the cornerstones of this administrative flexibility is a general remote mount model. The focus of this paper is to show how the features of this remote mount model can be used to simply and effectively present a single system image. We first describe some of the characteristics of the DS mount model, then describe the approach to single system image, and finally discuss some additional related topics.

Distributed Services Mount Model

Distributed Services uses "remote mounts" to achieve local/remote transparency. A remote mount is much like a conventional mount in the UNIX operating system, but the mounted filesystem is on a different machine than the mounted-on directory. Once the

remote mount is established, local and remote files appear in the same directory hierarchy, and, with minor exceptions, file system calls have the same effect regardless of whether files (directories) are local or remote.[1] Mounts, both conventional and remote, are typically made as part of system startup, and thus are established before users login. Additional remote mounts can be established during normal system operation, if desired.

Conventional mounts require that an entire file system be mounted. Distributed Services remote mounts allow mounts of subdirectories and individual files of a remote filesystem over a local directory or file, respectively. Fine granularity mounts are useful in configuring a single system image. For example, a shared copy of */etc/passwd* may be mounted over a local */etc/passwd* without hiding other, machine specific, files in the */etc* directory. Use of mounts at a fine granularity is key to this approach to single system image.

Virtual File Systems

The Distributed Services remote mount design is based on the Virtual File System approach used with NFS [5,6]. This approach allows construction of essentially arbitrary mount hierarchies, including mounting a local object over a remote object, mounting a remote object over a remote object, mounting an object more than once within the same hierarchy, mount hierarchies spanning more than one machine, etc. The main constraint is that mounts are only effective on the machine performing the mount.

In conjunction with using the Virtual File System concept, we necessarily have replaced the traditional *namei()* kernel function, which translated a full path name to an i-number, with a component by component *lookup()* function. *lookup()* is used both for local and remote path name resolution. The arguments to *lookup()* are a file *handle* representing a directory and the name of a component to be found in that directory. *lookup()* returns a handle for the component, if found. A handle is effectively a pointer to the on-disk inode for the corresponding object and a generation number for that inode. The generation number is used for subsequent validity tests.

When a client successfully requests a mount from a server, it receives a *handle* for the object it is mounting and stores it in its mount table. When the client is parsing a file path name, e.g., for *open()*, and encounters the mounted object, the handle is given to the server as an argument in the *lookup()* remote procedure call. Typically, the mounted object is a directory, and the server will look up an object within that directory.

For example, let us suppose that a client mounts server's */B* over */a/b*. The client then opens */a/b/c*. When the client gets to *b/c*, it passes the handle for *b* and the component *c* to the server, requesting the server to look up and return a handle for *c* that can be used in the actual *open()* call. The server will return a handle for */B/c*.

For fine granularity mounts, the string form of the file name component is returned, along with the file handle of the (real) parent directory. This alternative to using the file handle for the mounted file allows replacement of the mounted file with a new version without loss of access to the file (with that name). (For example, when */etc/passwd* is mounted and the *passwd* command is used, the old file is renamed *opasswd* and a new *passwd* file is produced. If we used a file handle for the fine granularity mount, then the client would continue to access the old version of the file. Our approach gives the effect, presumably intended, that the client sees the new version of the file.)

There are several points to notice here. First, this approach is stateless in that the server can be recycled (e.g., powered off and on) and the handle(s) given to the client(s) performing a mount(s) is still valid, so the mount need not be repeated. This is true because the handle refers to an on-disk structure, not an in-memory structure. Second, the path resolution process must necessarily ignore mounts on the server, since

1. The traditional prohibition of links across devices applies to remote mounts. In addition, Distributed Services does not support direct access to remote special files (devices) and the remote mapping of data files using the AIX extensions to the *shmat()* system call. Note that program licenses may not allow execution of a remotely stored copy of a program.

these are not reflected in the on-disk structures and are not necessarily repeated when the server is recycled. Third, as an immediate consequence, the client must explicitly perform all mounts "for itself," since it does not "see" mounts performed by the server.

Inherited Mounts

In constructing a single system image of UNIX systems, it is desirable, if not necessary, to preserve the traditional directory hierarchy and conventions. All the machines in the single system image must see the same instances of */etc/passwd*, */etc/hosts*, ..., home directories, spool directories for mail, and so forth. However, it is also desirable/necessary to be able to access local equivalents of these files/directories so that they may be kept up to date with the shared copies. For example, */etc/passwd* refers to a shared copy of the file, and */native/etc/passwd* refers to the unshared local version. In general */native/a/b/...* is established as the path to the local instance of */a/b/...*.

Without the concept of inherited mounts, discussed below, this implies that each machine would have to be doubly configured for its local (device) mounts. E.g., if / (root), */u*, and */usr* are on partitions /dev/hd0, /dev/hd1, and /dev/hd2, then the desired mounts could be achieved by the commands:

```
mount /dev/hd1 /u
mount /dev/hd2 /usr
mount / /native
mount /u /native/u
mount /usr /native/usr
```

Alternatively, the mount profile (*/etc/filesystems* in AIX) would contain an entry for each of these mounts. If another disk was added to hold */usr/src*, then two profile entries would be needed, one for mounting */usr/src* and one for mounting */native/usr/src*.

Distributed Services implements inherited mounts on top of virtual file systems. There is a *mntctl()* system call and corresponding remote procedure call. One of the options of *mntctl()* is to query and return a list of all mounts currently in effect on a given server. The *mount* command in AIX supports a `-i` (inherited) flag which causes the query to be performed and the additional mounts to be made. For the above example,

```
mount -i / /native
```

would have the same net effect as the three separate mount commands for the */native* subtree. When additional device mounts are configured, this single mount command still provides the desired effect of an aliased naming path for the local instance of the file hierarchy. Additional examples of motivation for inherited mounts are given in [3].

Presentation of Single System Image

Objectives

- The configuration is managed by a few simple profiles. It should be easy to add/delete machines and users, and to make other configuration changes.

- All of the machines in the single system image cluster should use exactly the same configuration files, i.e., there is no distinction between the profiles on the server for */etc/passwd* and related files and the ones on the clients.

- As a result of the above, it should be simple to reconfigure to use a different server for the */etc* files, either because of planned outages of the existing server or because of failure of the existing server.

- Client machines should recognize when the server is unavailable, and switch to alternate copies of administrative files and other shared files.

- Local replicated copies of the administrative files should be periodically updated, so that if there is an unplanned outage of the server, the other machines have up to date copies.

- If a machine providing some of the home directories is unavailable, a user should discover this immediately at login time, and and be able to either use an alternate home directory or wait until the machine becomes available again.

General Approach

The discussion will focus on administrative files, e.g., */etc/passwd*, and data files, e.g., home directory subhierarchies and mail spooling areas. Assuming, for the moment, a homogeneous processor architecture, executable files may be viewed between two extremes: (1) all of the machines in the cluster have full copies of the executable code, and so there is no sharing of executables, or (2) there is a single shared copy of each executable file. The first extreme has the potential for inconsistency among the multiple copies, administrative burden to ensure that inconsistent copies are not present, and waste of disk space for the redundant copies. The second extreme has the limitation that executables will be unusable if the shared copy is not accessible. An administrator will typically choose a policy in between the extremes, e.g., that the kernel and the files in */etc* and */bin* are replicated, but that other executables are shared. The approach discussed below supports and allows flexibility in determining such policies. However, AIX and DS have other administrative mechanisms, known as "code serving" which address these policies in detail, so we focus on the administrative and data files.

Where heterogeneous processor architectures are involved, the motivation for a separate mechanism for code serving is stronger, since the mechanisms below should not be used for sharing binary executables across different processor architectures. The files explicitly shared in the mechanisms described below are in ASCII format, and are suitable for sharing across heterogeneous processor architectures. Thus the mechanisms themselves will work across heterogeneous processor architectures. However, in the heterogeneous environment, machine boundaries are much more likely to be visible, e.g., due to byte order considerations in application data. More stringent requirements must be placed on application code in such an environment, if the illusion of a single system is to be preserved.

Configuration Files

/etc/adminserver. One machine is designated as the "administrative server" and is the machine that has the disk copies of the shared administrative files such as */etc/passwd*. The administrative server can be changed while machines are in operation, as discussed below. The name of the administrative server is stored in */etc/adminserver*.

/etc/SSImachines. This file lists the names of all machines in the single system image (including the administration server).

/etc/server.files. This file lists individual files that will be shared based on the administrative server's copy. For example, this list might include

```
/etc/passwd
/etc/group
/etc/motd
/etc/qconfig
                (AIX printer configuration file)
/etc/hosts
/etc/hosts.equiv
/etc/adminserver
/etc/SSImachines
/usr/adm/user.cfile
                (AIX adduser defaults)
/etc/server.files
/etc/server.dirs                (see below)
/etc/remounts.list              (see below)
/etc/ug.SSI
                (for id translations - see below)
/etc/oug.SSI
/etc/opasswd
/etc/ogroup
/etc/umountd.c
                (source for umountd - see below)
/usr/adm/newuser.sys
                (AIX adduser defaults)
/usr/adm/newuser.usr
```

Though the list could be longer or shorter, roughly this set of files has been appropriate in our experience.

/etc/server.dirs. This file lists directories, other than home directories, that will be shared based on the administrative server's copy. Assuming that code serving is handled separately, this list might include

```
/usr/mail
/usr/lib/news
/usr/spool/news
/usr/man
...
```

If code serving is not handled separately, then */usr/bin*, */usr/lib*, ... might be added to this list.

/etc/remounts.list. This file lists files and directories that will be unmounted when it is detected that the server is inaccessible and remounted when the server becomes accessible again. This is handled by the *umountd* daemon, discussed below. This list will be a subset of the combined lists in server.files and server.dirs, e.g.,

```
/etc/passwd
/etc/group
/etc/motd
/etc/qconfig
                (AIX printer configuration file)
/etc/hosts
/etc/hosts.equiv
/etc/adminserver
/etc/remounts.list
/usr/mail
```

This is a subset of the combined list oriented toward normal operation when the administrative server is inaccessible. It is a subset because some operations, e.g., changing passwords, presumably will be deferred when the administrative server is inaccessible, and some directories, e.g., */usr/man* and */usr/spool/news*, are likely to be empty, except on the administrative server, and thus uninteresting when that machine is unavailable.

/etc/ug.SSI, /etc/oug.SSI. DS provides general translation mechanisms for user and group id translation [1,2]. For machines within the cluster, there should be one to one translations, so that numeric id's are the same on all machines in the cluster, but machines in the cluster may also need translations to other machines outside the cluster. *ug.SSI* is used for a cluster wide definition of the translations. For brevity, we will not discuss the content of these files.

Home Directories

For the sake of simplicity, it is assumed that home directories' path names have the form .../machine/user, where "machine" is the name of the machine where the home directory is actually stored. Even though paths are of this form, users will see the same actual home directory on each machine of the cluster, e.g., in our environment, when user *sauer* is logged into machine *d75*, his home directory is still */u/auschs/sauer*, since his home directory is stored on *auschs*. This is a minor sacrifice of transparency, since users usually do not use rooted paths to get to their home directories – their home directory is listed in */etc/passwd*, so that is where they start, *cd* takes them back there, and the C shell ~ notation is often used to get to other users' home directories, e.g., *cd ~dale*. Shipley has proposed a similar convention for sharing home directories [7].

Having paths of this form allows each machine to simply mount the home directories stored on other machines, e.g.,

```
mount -n auschs /u/auschs /u/auschs
```

or, in general,

```
for i in 'cat /etc/SSImachines'
do
    if [ $i != $myname ]
    then
        mount -i -n $i /u/$i /u/$i
    fi
done
```

This is a slightly simplified fragment from */etc/SSImounts*, discussed below.

Machine Initialization

As init processing, after normal standalone initialization, e.g., *fsck* and device mounts, */etc/rc* starts DS and then runs */etc/SSImounts*. *SSImounts* runs in the background so that local operations can begin without server availability. *SSImounts* runs on all machines, including the adminserver. Following are simplified sketches taken from *SSImounts*. Error checks, *touch*es and *mkdir*s for mount points, precautionary *umounts*, etc. are omitted.

Initial mounts from adminserver, updating local copies of files:

```
if [ $myname != $adminserver ]
then
    until mount -i -n $adminserver \
        /native /$adminserver
    do
        sleep $delay
    done
    for i in 'cat /etc/server.files'
    do
        cp -p /$adminserver$i $i
        mount -n $adminserver /native$i $i
    done
    for i in 'cat /etc/server.dirs'
    do
        mount -i -n $adminserver /native$i $i
        if [ $i = '/usr/mail' ]
        then
            /etc/movemail & # see below
        fi
    done
fi
```

Start */etc/umountd*

```
make umountd
/etc/umountd &
```

Update user/group ids.

```
cmp /etc/ug.SSI /etc/oug.SSI
if [ $? -ne 0 ]
then
    dsldxprof -a -f /etc/ug.SSI
    if [ $? -eq 0 ]
    then
        cp -p /etc/ug.SSI /etc/oug.SSI
    fi
fi
```

Mount home directories. This is as indicated in the previous fragment, except that the mounts are retried asynchronously in the background so that availability of any given machine doesn't delay availability of home directories from other machines.

Once these steps have been performed, then the machine has joined the single system image.

/etc/movemail. Mail received while the administrative server is not available will be kept in the native spool directory, */usr/mail*. *movemail* moves mail from the native spool directory to the shared spool directory whenever the shared directory is mounted, either by *SSImounts* or *remounts* (see below).

umountd

The key remaining topic is the daemon *umountd*. *umountd* uses a polling loop, performing the following functions and then sleeping until repeating the functions. The default sleep time is 60 seconds.

Detection of server inaccessibility. umountd attempts to open each of the files listed in */etc/server.files*. If an open fails, *umountd* assumes the server is inaccessible and executes */etc/remounts*. */etc/remounts* unmounts all of the files in */etc/remounts.list* and then attempts to remount them. *remounts* will execute *movemail* after successfully remounting */usr/mail*. (*umountd* runs on adminserver, but skips these steps.)

Updating modified files. umountd determines whether any of the files in */etc/server.files* have been updated. If so, *umountd* locks the server copy and updates the native copy. (*umountd* running on adminserver skips these steps.)

Detection of configuration changes. If key configuration files, e.g., */etc/adminserver* or */etc/server.files*, have been updated, *umountd* execs *SSImounts*. *SSImounts* applies the changes and then starts a fresh version of *umountd*, as indicated above.

There are subtleties of locking and timing which we omit for brevity. Starting with DS 1.2, the source for *umountd.c* is included in the DS samples directory, along with the installation documentation, installation command and so forth.

Note the power of the above mechanisms. By simply changing the name of the server in */etc/adminserver*, e.g., for a scheduled outage of the adminserver, the machines in the cluster will shift to the new adminserver in a couple of minutes, without rebooting any of the machines or otherwise significantly disrupting users. For an unscheduled server outage of significant duration, a switchover to a different server can be accomplished by changing the adminserver file on each of the other machines and rebooting them. In either case, scheduled or unscheduled, the original server can rejoin the cluster as a client when it is ready to rejoin the cluster, and then assume the server role again when its configuration files have been updated.

Summary

We believe this approach effectively meets the operational definition given in the introduction, in regard to user accounts, data, availability, and administered services. The mechanisms are designed to be simple to apply and administer, yet highly effective in presenting the image of a single system. These mechanisms are complementary to the underlying file system mechanisms of Distributed Services, and orthogonal to other enhancements such as the code server mechanisms. Thus the same underlying mechanisms are applied across the distributed environment, for example the abstraction of our software development environment depicted in Figure 1. In addition, significant administrative flexibility is present, as suggested in the preceding paragraph. These concepts could be applied to other distributed file systems supporting fine granularity mounts.

References

1. Charles H. Sauer, Don W. Johnson, Larry Loucks, Amal A. Shaheen-Gouda, and Todd A. Smith, "RT PC Distributed Services: Overview," *Operating Systems Review 21*, 3 (July 1987) pp. 18-29.
2. Charles H. Sauer, Don W. Johnson, Larry Loucks, Amal A. Shaheen-Gouda, and Todd A. Smith, "RT PC Distributed Services: File System," *;login: 12*, 5 (September/October 1987) pp. 12-22.
3. Charles H. Sauer, Don W. Johnson, Larry Loucks, Amal A. Shaheen-Gouda, Todd A. Smith, "Statelessness and Statefulness in Distributed Services," UniForum 1988, Dallas, TX, February 1988.
4. Jason Levitt, "The IBM RT Gets Connected," *Byte 12*, 12 (1987) pp. 133-138.
5. R. Sandberg, D. Goldberg, S. Kleiman, Dan Walsh and B. Lyon, "Design and Implementation of the Sun Network File System," USENIX Conference Proceedings, Portland, June 1985.
6. Sun Microsystems, Inc., *Networking on the Sun Workstation*, February 1986.
7. M. Shipley, "The Virtual Home Environment," UniForum 1988, Dallas, TX, February 1988.

C++ Tape

The first 1988 USENIX software tape contains C++ software. It requires no AT&T nor UC license. It is being sent to all current Institutional and Supporting members of the Association.

Individual members of USENIX who wish to obtain a copy of the tape may request it from the Association Office, which will then send the requestor a "Tape Release Form." The form plus a check for $125 should be returned to the office, whereupon the tape will be sent out, postpaid in the US. Foreign individuals will be billed for the additional (air) postage/shipping.

The tape, in *tar* format at 1600 BPI, contains:

GNU C++ Version 1.21.0 (Michael Tiemann)
OOPS (Keith E. Gorlen)
Storage management class and String class... (Peter A. Buhr)
InterViews 2.3 (Mark A. Linton)
C++ Subroutines for string manipulation (Arthur Zemon)

The tape is not available to non-members of the USENIX Association.

* * *

If you are were an institutional member in 1987 and have not received your 1987 software distribution tapes, please contact the office.

2.10BSD Software Release

The "Second Berkeley Software Distribution" (2.10BSD), produced by the Computer Systems Research Group (CSRG) of the University of California, Berkeley, is being distributed by the USENIX Association. It is available to all V7, System III, System V, and 2.9BSD licensees for a price of $200. The release consists of two 2400 foot, 1600 BPI tapes (approximately 80Mb) and approximately 100 pages of documentation. If you require 800 BPI tapes, please contact USENIX for more information.

Sites wishing to run 2.10BSD should be aware that the networking is only lightly tested, and that certain hardware has yet to be ported. Contact Keith Bostic at the address below for current information as to the status of the networking. As of August 6, 1987, the complete 4.3BSD networking is in place and running, albeit with minor problems. The holdup is that only the Interlan Ethernet driver has been ported, as well as some major space constraints. Note, if we decide to go to a supervisor space networking, 2.10 networking will only run on:

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

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

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

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

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

{ucbvax,seismo}!keith
keith@okeeffe.berkeley.edu

+1 415 642-4948

Keith Bostic
Casey Leedom

UUNET Communications Service

The UUNET Communications Service is now one year old and has proven itself to be a success. UUNET is a non-profit communications service that provides access to USENET news, UUCP mail, public domain software, and many standards. The UUNET computer is accessible by any computer running the UNIX operating system. With the purchase of a third party software package, any PC running MS-DOS can also access UUNET.

The UUNET Communications Service came about because many people were having difficulty in accessing USENET, the worldwide network of UNIX users. When a USENET connection was found, it was often expensive to maintain, unreliable, or prone to unanticipated interruption when the remote computer was used for work rather than for USENET access. The USENIX Association funded the UUNET Communications Service to assist UNIX users in accessing the USENET network and in sending electronic mail to other UNIX users.

UUNET, because of its mandate, has become the best connected UNIX computer in the world. It provides its subscribers with easy access to the USENET news and mail network and to other UNIX users. UUNET has been authorized to function as an ARPANET mail gateway. Gateways to other networks are being set up. UUNET is also the principal gateway to European, Australian, Asian, and South American UUCP sites.

UUNET provides subscribers with UUCP access to an extensive archive of publicly available UNIX software. This includes the latest GNU software, all the ARPANET RFCs, the latest UUCP map information, the latest Kermit distributions, and the netlibd archives.

Operationally, UUNET consists of a 16-processor Sequent B21 computer with 20 dialup modems (12 accessible via an 800 number), a T-1 connection, over one gigabyte of disk space, and a 56 KBPS dedicated connection to Tymnet. This system is dedicated to UUNET and has no function other than as a communications relay. Most subscribers access UUNET through the Tymnet X.25 public data network or via Telebit Trailblazer Plus modems.

Because the UUNET Communications Service is non-profit, costs to subscribers can be kept very low. Most subscribers call in at night, taking advantage of the $4.00 per hour Tymnet off-peak rate.

UUNET currently has over 360 subscribers, including individuals, universities, computer companies, banking and investment firms, and other companies. Many subscribers save a substantial sum of money by using UUNET rather than paying long distance charges to receive news and mail.

For further information on the UUNET, provide your name and US postal address to the UUNET office at:

{ucbvax,usenix}!uunet!uunet-request

(703) 764-9789

UUNET Communications
P.O. Box 2685
Fairfax, VA 22031

;login:

The USENIX Association Newsletter

Volume 13, Number 5 **September/October 1988**

CONTENTS

The closing date for submissions for the next issue of *;login:* is October 28, 1988

USENIX THE PROFESSIONAL AND TECHNICAL UNIX® ASSOCIATION

C++ Conference Program

Denver Marriott, City Center Hotel
Denver, Colorado

October 17-20, 1988

USENIX is holding its first full C++ conference in Denver, Colorado, Monday through Thursday, October 17-20.

Program At A Glance / Dates to Remember

- Hotel Registration Deadline — September 26
- Pre-registration Deadline — September 28
- All Day Tutorials — October 17-18
- Technical Sessions — October 19-20

For details about registration, contact the USENIX Conference Office immediately.

C++ Tutoral Program

M1: Advanced C++ Topics — Monday, 9-5:00
Jonathan Shopiro, AT&T Bell Laboratories

M2: Object-Oriented Development and C++ Concepts — Monday, 9-5:00
John Carolan and Adrienne Dockrell, Glockenspiel Ltd.

T1: An Introduction to C++ — Tuesday, 9-5:00
Robert Murray, AT&T Bell Laboratories

T2: Applications: InterViews & C++ 2.0 — Tuesday, 9-5:00
Part A: Programming User Interfaces in C++ With InterViews
Mark A. Linton, Stanford University
Part B: What's New in C++ 2.0
Stanley Lippman, AT&T Laboratories

Opening Session — Wednesday, 9-10:30

Keynote speech
W. N. Joy, Sun Microsystems

Parameterized Types for C++
Bjarne Stroustrup, AT&T Bell Laboratories

Technique — Wednesday, 11-12:30

Building Well-behaved Type Relationships in C++
R. B. Murray, AT&T Bell Laboratories

Porting from Common Lisp with Flavors to C++
Joseph Eccles, AT&T Bell Laboratories

Databases and File Systems — Wednesday, 2-3:30

Prototyping Database Applications with a Hybrid of C++ and a 4GL
Ronan Stokes, Glockenspiel, Ltd.

Open Dialogue: Using an Extensible Retained Object Workspace to Support a UIMS
Andrew Schulert, Kate Erf, Apollo Computer

Building Object-Oriented UNIX-like File Systems in C++
Peter Madany, Doug Leyens, Vince Russo,
Roy Campbell, University of Illinois

Applications

Wednesday, 4-5:30

Applying Object-Oriented Design to Structured Graphics
John M. Vlissides, Mark A. Linton, Stanford University

A C++ Interpreter for Scheme
Vincent F. Russo, Simon M. Kaplan, University of Illinois

GPIO: Extensible Objects for Electronic Design Tools
David Campbell, Russel Edwards, Prakash Reddy,
Roger Scott, Data General Corporation

Experience

Thursday, 9-10:30

C++: From Research to Practice
S. B. Lippman, B. E. Moo, AT&T Bell Laboratories

NAPS – A C++ Project Case Study
C. Berman, R. Gur, AT&T

Parallelism and Simulation

Thursday, 11-12:30

Data-Level Parallel Programming in C++
Thomas M. Breuel, MIT

A Multiprocessor Operating System Simulator
Gary M. Johnston, Roy H. Campbell, University of Illinois

Modelling of Control Systems with C++ and PHIGS
Dag M. Brück, Lund Institute of Technology

Linguistics

Thursday, 2-3:30

Type-safe Linkage for C++
Bjarne Stroustrup, AT&T Bell Laboratories

Implementing a Logic-Based Executable Specification Language in C++
Peter A. Kirslis, Robert B. Terwilliger, AT&T Bell Laboratories

Debugging and Instrumentation of C++ Programs
Martin J. O'Riordan, Glockenspiel, Ltd.

Libraries

Thursday, 4-5:30

libg++, The GNU C++ Library
Douglas Lea, SUNY Oswego

A Class Library for Darts
Troy Otillo, Cal Poly - SLO

Guide to the C++ Real Library
Jerry Schwarz, AT&T Bell Laboratories

Iris: A Class-Based Window Library
E. R. Gansner, AT&T Bell Laboratories

Call for Papers
Winter 1989 USENIX Conference
San Diego, California
January 30 - February 3, 1989

Papers are requested for formal review as candidates for inclusion in the three day technical session at the 1989 Winter USENIX Conference. Papers that are accepted will be presented at the conference and published in the conference proceedings. The technical sessions provide a forum for the presentation of new research and development related to or based upon the UNIX operating system.

Suggested topics include (but are not limited to):

- Performance analysis and tuning
- New User Interfaces and Applications
- System and Network Security
- Networking and Distributed Services
- RISC versus CISC in UNIX
- Software and System Management tools
- Standards
- Graphics and Electronic Publishing
- Evolution of UNIX for the 1990's

All papers should describe new and interesting work. Acceptance or rejection of a paper will based completely on the work submitted at the deadline. Submitted papers should consist of a 100 to 300 word abstract in addition to the main body of the paper. Extended abstracts will be conditionally accepted but full papers are preferred. Papers accepted on extended abstracts that do not meet the promise of the abstract will be rejected. Each paper should discuss how this work relates to prior work and provide sufficient detail in the presentation of background material and work to allow referees to perform a consistent comparison to other submitted papers. Concise references of related work should also be included as appropriate. Full papers should be 6-12 single spaced typeset pages and include any abstract, references, or illustrations. For the review process you should submit the highest quality copy you can create. Laser printer output is recommended. The exact format for final papers will be sent to authors of accepted papers.

Four hard copies and one electronic copy of each submitted paper must arrive no later than **October 7, 1988**; this is an absolute deadline. Papers received after that date will not be considered. Papers which clearly do not meet USENIX's standards for applicability, originality, completeness or page length may be rejected with no review. Authors will receive official notification no later than **November 4, 1988**, and final papers are due by **December 5, 1988**.

Please contact one of the program chairs if additional information is required:

Greg Hidley, (619) 534-6170
Keith Muller, (619) 534-4062

They may be also be reached at *sdusenix@ucsd.edu*.

Send technical program submissions to:

Greg Hidley
CSE Dept. C-014
University of California, San Diego
La Jolla, CA 92093

Bitnet: sdusenix@ucsd
Internet: sdusenix@ucsd.edu
uucp: ucbvax!ucsd!sdusenix

The program committee includes:

Rick Adams	Center for Seismic Studies
Keith Bostic	UC Berkeley
Todd Brunhoff	Tektronix
John Chambers	University of Texas, Austin
Lori Grob	NYU
Andrew Hume	AT&T Bell Labs
Thomas Narten	Purdue University
Don Seeley	University of Utah
Melinda Shore	Frederick Cancer Research Facility
Gene Spafford	Purdue University
Henry Spencer	University of Toronto
Avadis Tevanian	NeXT
Karen White	Pyramid

Workshop on Large Installation Systems Administration

DoubleTree Hotel
Monterey, California

November 17-18, 1988

In light of last year's successful workshop on Large Installation Systems Administration, Alix Vasilatos will again be chairing a workshop on this subject in Monterey, CA on Thursday and Friday, November 17th and 18th, 1988. There is demonstrable benefit in bringing together system administrators of sites with 100 or more users (on one or more processors) to compare notes on solutions that they have found for a variety of common problems. These include but are not limited to:

Large file systems (dumps, networked file systems)
Password file administration
Large mail system administration
USENET/News/Notes administration
Heterogeneous environments (mixed vendor and/or version)
Load control and batch systems
Monitoring tools
Software release to multiple systems
Output device management

The workshop will focus on short papers and presentations. You do not have to present to attend! Proceedings will be available at the workshop.

Rob Kolstad of Prisma Computers will be the Keynote Speaker. His topic will be "The Evolving Role of the System Administrator."

For details about registration, contact the USENIX Conference Office at (213) 592-3243 or *{uunet,ucbvax}!usenix!judy*.

Call for Papers
EUUG Spring '89 Conference

Brussels, Belgium
April 10-14, 1989

The BUUG will host the Spring '89 European UNIX systems User Group Technical Conference in Brussels. Technical tutorials on UNIX and closely related subjects will be held on Monday and Tuesday, followed by the three day conference with commercial exhibitions. A pre-conference registration pack will be mailed to interested persons in early December.

Call for Papers

The EUUG invite abstracts from those wishing to present their work. Submissions from students are particularly encouraged under the EUUG Student Encouragement Scheme, details of which are available from the Secretariat. All submitted papers will be refereed to be judged with respect to their quality, originality and relevance. Abstracts MUST be submitted by post to the EUUG Secretariat. All submissions will be acknowledged. Suggested subject areas include:

- real time
- networking
- security issues
- graphics
- internationalisation
- distributed processing
- fault tolerance
- new architectures
- transaction processing
- window systems and environments
- supercomputing
- standards and conformance tests

Important Dates:

Abstract deadline	November 30, 1988
Acceptance notification	January 15, 1989
Final paper received	February 1, 1989

Tutorial Solicitation

Tutorials are an important part of the EUUG's biannual events, providing detailed coverage of a number of topics. Past tutorials have been taught by leading experts. Those interested in offering a tutorial should contact the EUUG Tutorial Officer as soon as possible.

Additional Information

The Programme Chair will be pleased to provide advice to potential speakers.

If you wish to receive a personal copy of further information about this, and future, EUUG events, please contact the Secretariat.

Secretariat

EUUG
Owles Hall
Owles Lane
Buntingford
Herts, SG9 9PL, UK

Phone: +44 763 73039
Fax: +44 763 73255
Telex:
Email: euug@inset.uucp

Tutorial Officer

Neil Todd
IST
60 Albert Court
Prince Consort Road
London, SW7 2BH, UK

+44 1 581 8155
+44 1 581 5147
928476 ISTECH G
neil@ist.co.uk

Programme Chair

Prof. Marc Nyssen
Medical Informaticas Dept.
Vrije Universiteit Brussel
Laarbeeklaan 103
B-1090 Jette Belgium

+32 2 477 44 24
+32 2 477 40 00

marc@minf.vub.uucp

Obtaining GNU Software

The GNU Project (GNU's Not UNIX) is developing a complete UNIX compatible software system with freely redistributable source code. The rationale for GNU is explained in the GNU Manifesto. Copies are available in the GNU Emacs distribution and manual, and by request to *gnu@prep.ai.mit.edu*.

You are encouraged to get GNU software from or with others. GNU software is also available by *ftp* on the DoD/NSF Internet, and by *uucp*; ask *gnu@prep* for details. If you are unable to use one of these methods, you can use this order form.

___ at $ 150 = $ _____	**GNU Emacs source code and other software,** on 1600bpi tape in *tar* format. The tape also includes Scheme, T, Hack, Bison, GNU Chess, GDB, and the X window system (Version 10r4).
___ at $ 150 = $ _____	**GNU C Compiler.** Beta test tape, 1600bpi, in *tar* format. The tape also includes Bison, Gawk, GNU Assembler, X windows (Version 11r2 complete), Flex, GNU Make, and object file utilities.
___ at $ 175 = $ _____	**GNU Emacs source and other software,** cartridge tape for Suns.
___ at $ 175 = $ _____	**GNU C Compiler and other software,** cartridge tape for Suns.
___ at $ 150 = $ _____	**GNU Emacs source and binary code,** on 1600bpi tape in VMS interchange format.
___ at $ 150 = $ _____	**GNU C Compiler source and binary code,** on 1600bpi tape in VMS interchange format.
___ at $ 15 = $ _____	**GNU Emacs manual,** ~300 pages, spiral bound. The manual is phototypeset and offset printed, and includes a reference card.
___ at $ 60 = $ _____	**Box of six** GNU Emacs manuals, each with reference card.
___ at $ 1 = $ _____	**GNU Emacs reference card.**
___ at $ 5 = $ _____	**Packet of ten** GNU Emacs reference cards.
___ at $ 10 = $ _____	**GDB manual,** ~50 pages, side stapled.
___ at $ 10 = $ _____	**Texinfo manual,** ~100 pages, side stapled.
___ at $ 10 = $ _____	**Termcap manual,** ~60 pages, side stapled.
Sub Total $ _____	
$ _____	5% Massachusetts sales tax, if applicable.
$ _____	Optional tax deductible donation.
$ _____	Outside North America and Hawaii, add $15 for *each* tape or manual for shipping costs; and add $60 for *each* box of manuals.
Total $ _____	

Prices are subject to change without notice. All software from the Free Software Foundation is provided on an "as is" basis, with no warranty of any kind. TeX source for all manuals is on the appropriate distribution tape. Many of these programs are covered by a General Public License that permits everyone to have and run copies of them, at no charge, and to redistribute copies under certain conditions which are designed to make sure that that all modified versions of the program remain as free as the versions we distribute. The General Public License is usually in a file named *COPYING*.

Orders are filled upon receipt of check or money order. We do not have the staff to handle the billing of unpaid orders. Please help keep our lives simple by including your payment with your order. Make checks payable to "Free Software Foundation," and mail orders to:

Free Software Foundation
675 Mass. Avenue
Cambridge, MA 02139
+1 (617) 876-3296

Name: _____
Organization: _____
Address: _____
City, State, Zip: _____

The USENIX Association is printing the above as a service to the user community; no endorsement of GNU software is implied.

Broadcast Storms, Nervous Hosts, and Load Imbalances

Paul E. McKenney

Information Sciences and Technology Center
SRI International

ABSTRACT

As equipment connected to local-area networks becomes more complex and diverse, so do possible modes of failure. Innocent-looking "features" of such equipment can easily cause serious disruptions of service to a local-area network.

Some software problems (including broadcast storms, "nervous hosts," and load imbalances) that can arise on Ethernets using the DDN protocol suite are examined herein. The causes of these problems and the sequence of events leading up to them are explored in detail, and some commonly known methods of preventing, diagnosing, and correcting the problems are presented.

Overview

This paper will look at three of the types of problems which afflict local-area networks:

- Broadcast storms
- Nervous hosts
- Load imbalances.

The following sections will look at these problems in some detail, describing their causes, how they can be diagnosed, and how they can be prevented or corrected. Since the broadcast storm is the most complex and the most damaging of these problems, it will be described in considerably more detail.

Broadcast Storms

While a classic broadcast storm occurs when a large number of hosts respond almost simultaneously to a broadcast packet, the effects of a broadcast storm (nearly total denial of network services for an extended period) can be induced by a number of mechanisms. The term "broadcast storm" will used in this more general sense for the rest of this paper.

Broadcast storms can be caused by continual packet transmission and by inappropriate responses to broadcast packets.

Continual Packet Transmission

Since Ethernet is multiple-access, there is nothing to prevent a host from consuming most of an Ethernet cable's bandwidth by continually transmitting garbage packets. This extreme case can be remedied only by either fixing the software or physically disconnecting the host from the Ethernet.

Note that bugs that cause continual packet transmission can occur in user software just as easily as they can in the kernel. A normal user program on a Sun-3 work station can easily send well over 100 packets per second through a UDP socket. As few as five or ten programs doing this simultaneously on separate work stations (e.g., *rwhod*[1]) can do a very good job of congesting an Ethernet.

The Ethernet source address is relatively immune to corruption by software bugs because it is usually stored in a hardware register in the Ethernet interface.[2] Therefore, keeping a list of the Ethernet addresses of all machines on a network can help pinpoint the

[1] This program periodically broadcasts the list of users currently logged onto the host that it is running on. While this allows users to easily see who is logged onto other machines, it can also produce bursts of heavy traffic.

[2] The Ethernet source address is not always completely immune to corruption. For example, hosts running DECNET set their Ethernet addresses to a value related to their DECNET node ID under software control.

source of garbage packets. This pinpointing is accomplished by extracting the Ethernet source address from the garbage packet (possibly using Van Jacobson's *tcpdump* program), then looking up the source address in the list to find the offending host. For those who do not wish to maintain such a list manually, we at SRI have written an "etherhostprobe" program that makes a list of Internet addresses and Ethernet addresses of all hosts connected to the local Ethernet that implement address resolution protocol (ARP).

Responding Inappropriately to Broadcast Packets

The classic broadcast storm ensues when several hosts attempt to forward the same datagram over the broadcast medium it came from.

A classic broadcast storm involves Internet protocol (IP) packets and ARP packets. IP packets are the building blocks for the familiar transmission control protocol (TCP), network file system (NFS), and remote procedure call (RPC). ARP packets allow hosts to associate Internet addresses with the corresponding Ethernet addresses.

The following sections will describe a classic broadcast storm in more detail by looking first at the formats of the IP and ARP packets, second at the IP forwarding mechanism, and finally at a (small) example of a classic broadcast storm. This will be followed by some experimental results obtained at SRI and by recommendations for preventing classic broadcast storms.

Ethernet Header Format

All packets on an Ethernet (including IP and ARP packets) have an Ethernet header prefixed to them as follows:

- Six-byte source address
- Six-byte destination address
- Two-byte packet type.

An Ethernet address is typically written in hexadecimal form, separated by colons, e.g., `1c:08:1e:3d:00:0a`. The Ethernet broadcast address is `ff:ff:ff:ff:ff:ff`; a packet with this as its destination address will be received by all local Ethernet interfaces.

Each Ethernet interface ignores all packets except those whose destination address is either that Ethernet interface's address or the broadcast address. Ethernet packets destined for the broadcast address form the seeds of a broadcast storm.

An Ethernet packet type of **0800** (hexadecimal) indicates that the packet is an IP packet; an Ethernet packet type of **0806** (hexadecimal) indicates that the packet is an ARP packet.

IP Header Format

An IP packet has many fields, but the only one relevant to this discussion is the four-byte IP destination address (see Request For Comments (RFC) 791 for more details). An IP address is typically written in dotted-decimal form, e.g., `10.0.0.51`. Each IP address is divided into a "network part" and a "host part" with the latter part broken down further in a subnetted network (see RFCs 922, 950, and 1027 for details). Table 1 shows how the different classes of nonsubnetted IP addresses are decomposed into network and host parts.

First Byte		Class	Network	Host
0	127	A	1	3
128	191	B	2	2
192	223	C	3	1
224	239	D		
240	255	E		

Table 1: IP Address Classes

Class D addresses are special multicast addresses (see RFCs 966 and 988), and Class E addresses are reserved for future use. Only Class A, B, and C addresses are relevant to this discussion. (See RFCs 1010 and 1020 for more details on IP address assignment.)

If the host part of the IP address is composed of all 255s (for example, `10.255.255.255`), the IP packet is to be broadcast to all hosts on network 10.[3] Unfortunately, this convention was established

[3] However, since network 10 does not support the notion of broadcast, this IP packet would be ignored, although an error might be returned via an ICMP packet.

only fairly recently, so that some older implementations (in particular BSD 4.2) employ the convention that a host part composed of all 0s (e.g., 10.0.0.0) indicates that the IP packet is to be broadcast to all hosts on the network. As will be seen, this historical incompatibility constitutes the trigger that sets off the classical broadcast storm.

A sample new-style IP broadcast packet layered over Ethernet is shown in Table 2. The "IP data" would consist of the headers and data of the higher level protocol (e.g., user datagram protocol, or "UDP") that is layered on top of IP in this packet.

ARP Packet Format

The following fields from an ARP packet are relevant to this discussion (see RFC 826 for more details):

- Sender's Ethernet address
- Sender's IP address
- Target's IP address.

A typical ARP packet might be as as shown in Table 3. Here the host (call it Host A) whose IP address is 193.30.20.10 wants to know the Ethernet address for the host (call it Host B) whose IP address is 193.30.20.11. Since Host A does not know Host B's Ethernet address, the Ethernet destination address is set to the broadcast address to so that Host B can receive the packet. When Host B does receive the packet, it will note that the target IP address in the packet matches his own IP address, and will send a reply. Any other host that receives the packet will see that the target IP address in the packet does not match his own, and so will discard the packet.

IP Packet Forwarding

While an Ethernet interface can simply ignore packets that are not addressed to it, a host may be required to process IP packets that are addressed to someone else. For example:

- A host that is acting as a gateway between two networks must forward packets between the networks.
- A host that cannot forward a packet may inform the packet's originator by means of an

	Field	Value
Ethernet Header	Source Address	1c:08:1e:3d:00:0a
	Destination Address	ff:ff:ff:ff:ff:ff
	Packet Type	0800
IP Header	Destination IP Address	193.30.20.255
IP Data		

Table 2: New-Style IP Broadcast Packet

	Field	Value
Ethernet Header	Source Address	1c:08:1e:3d:00:0a
	Destination Address	ff:ff:ff:ff:ff:ff
	Packet Type	0806
ARP Packet	Sender's Ethernet Address	1c:08:1e:3d:00:0a
	Sender's IP Address	193.30.20.10
	Target's IP Address	193.30.20.11

Table 3: ARP Packet

Internet control message protocol (ICMP) packet.

A host must be very careful with packets that are not addressed to it. (See RFC 1009 for a fairly thorough discussion and RFCs 1015 and 1017 for some recent thoughts.)

A host must be even more careful with packets that have been broadcast, as they may have been received by many other hosts. A good principle is to never do anything with a broadcast packet unless a hundred and one hosts can do it at the same time safely on the same Ethernet.[4] Broadcast storms can occur when this principle is violated.

Both BSD 4.2 and BSD 4.3 violate this principle by refusing to check that the Ethernet destination address is the same as the Ethernet broadcast address (`ff:ff:ff:ff:ff:ff`). Both BSD 4.2 and BSD 4.3 do check the *Internet* destination address to determine whether it matches their idea of the *Internet* broadcast address (in fact BSD 4.3 checks the Internet destination address to see whether it matches any of the currently known Internet broadcast addresses), but there is no guarantee that

- It will not be necessary to add yet another Internet broadcast address at some time in the future, or that
- Some bug that wraps a broadcast Ethernet header around a single-destination Internet packet will not crop up somewhere.

Any check of the link layer (Ethernet in this case) broadcast address must be done at that layer; the results of the check must be available to the network layer (IP in this case), so that the layer interface would have to be modified to implement this check. This could go a long way toward explaining any reluctance the BSD 4.3 maintainers might feel about making this sort of modification.

There is a work-around for BSD 4.2 and BSD 4.3 systems in the form of a kernel variable called *ipforwarding*. Setting this variable to 0 will prevent any forwarding of IP packets. This has the (possibly unfortunate) side effect of making the system incapable of acting as a gateway between two networks or subnets. Stock systems have this variable set to 1 by default (thus *enabling* IP forwarding), although a BSD 4.3 system with a single network interface will behave as though it were set to 0 (thus *disabling* IP forwarding).

[4] Some examples of safe actions would be recording the packet on a local disk, responding to the packet if no other host is going to (e.g., if you are the only boot server, then you may respond to broadcast requests for booting), and, of course, discarding the packet.

A Classical Broadcast Storm

We now have the background to examine an example broadcast storm. Consider a small network with a single BSD 4.3 machine (`193.30.20.10`) whose IP broadcast address has been set to the standard `193.30.20.255`, and three BSD 4.2 machines, all of which have *ipforwarding* set to 1. Referring to Figure 1 we see the following sequence of events:

1. At time=0, host `193.30.20.10` broadcasts an IP packet over the network. This packet might look like the one in Table 2.

2. BSD 4.2 hosts `193.30.20.11`, `193.30.20.12`, and `193.30.20.13` all receive this packet.

3. Since the IP destination address `193.30.20.255` does not match either `193.30.20.0` (the BSD 4.2 broadcast address) or the hosts' own IP addresses, each host decides to forward the packet.

4. Each host looks in its cache of IP-address-to-Ethernet-address translations for the Ethernet address corresponding to `193.30.20.255`. Since there is no such host, the lookup will fail. Each host will therefore broadcast an ARP packet over the Ethernet at time=10 in an attempt to find the Ethernet address corresponding to `193.30.20.255`. Table 4 shows what this packet might look like for host `193.30.20.11`. The three simultaneous ARP packets collide and are therefore lost. Each of the hosts detects the collision, and schedules a retransmission at some random time in the future.

5. At time=30, hosts `193.30.20.11` and `193.30.20.13` retransmit. Again the ARP packets collide, are lost, and the hosts schedule a random retransmission.

	Field	Value
Ethernet Header	Source Address	1c:08:1e:3d:00:0b
	Destination Address	ff:ff:ff:ff:ff:ff
	Packet Type	0806
ARP Packet	Sender's Ethernet Address	1c:08:1e:3d:00:0b
	Sender's IP Address	193.30.20.11
	Target's IP Address	193.30.20.255

Table 4: Broadcast Storm ARP Packet

6. This time, all three hosts pick different times to retransmit their ARP packets. Since a collision is therefore avoided, the packet is broadcast successfully to all other hosts on the Ethernet.

7. Since there is no actual host with an IP address of `193.30.20.255`, there is no reply to the ARP packets. Thus, all three of the hosts receiving the original broadcast IP packet will time out and drop the packet.

The net effect is that one broadcast packet has caused no fewer than eight separate transmissions.

We at SRI conducted larger-scale tests of the classical-broadcast-storm mechanism on our local Ethernet (after hours, of course), which is populated by a large diverse population of hosts, including over one hundred Sun workstations, several Vaxes running BSD 4.3, several more Vaxes running VMS, some Xerox workstations, several IBM PCs, several UNIX machines, and an IBM mainframe.

The broadcast storm was triggered by a burst of *rwhod* packets broadcast on the new-style (all 1s) broadcast address by a Silicon Graphics work station. The first set of trials used a total of 43 *ipforwarding* hosts; each trial generated over 400 packets per second for a period of twenty seconds.

The second set of trials used a total of 23 *ipforwarding* hosts; each of these trials generated over 400 packets per second for a period of ten seconds. Decreasing the number of *ipforwarding* hosts shortens the duration of the broadcast storm commensurately.

The packet rate figures are subject to large errors because of missed interrupts in the measuring hosts and to high collision rates on the Ethernet. The figures given above almost certainly understate the actual packet rates.

The standard measures for preventing classical broadcast storms are well known, but bear repeating:

- Clear the *ipforwarding* kernel variable in each UNIX host on the network, or
- Upgrade each system to a version that has BSD 4.3 networking.

If you have a BSD 4.3 system with more than one network interface, it is still wise to clear the *ipforwarding* kernel variable (unless is is really needed as a gateway).

Most BSD 4.2 and BSD 4.3 systems can be patched with *adb* to clear the *ipforwarding* variable as follows:

```
cp  /vmunix  /vmunix.save
adb  -w  /vmunix
_ipforwarding?*W  0
$q
/etc/reboot
```

Note that the change will not take effect until after the reboot.

SRI experimented with an *ipforwarding* program that can determine whether a host will forward new-style IP broadcast packets. This program must be run on a Sun[5] that is connected to the same subnet as the host to be tested. This program works by handcrafting a "tickler" IP packet with a single-destination Ethernet header and an new-style broadcast IP header (see Table 5). After sending this packet, the program listens for an ARP packet requesting the Ethernet address that corresponds to the new-style IP broadcast

[5] The program uses Sun's network interface tap (NIT) to send and receive raw Ethernet packets.

	Field	Value
Ethernet Header	Source Address	1c:08:1e:3d:00:0a
	Destination Address	1c:08:1e:3d:00:0b
	Packet Type	0800
IP Header	Destination IP Address	193.30.20.255

Table 5: IP-Forwarding "Tickler" Packet

address. If such a packet is heard, this means that the destination host does forward new-style broadcast IP packets.

Note that this program does not induce a broadcast storm, since it "tickles" hosts one at a time.

Other Packet-Response Storms

There are other mechanisms that theoretically can result in a packet-response storm:

- Many reverse ARP (RARP) servers serving the same Ethernet address
- Many network mask request servers serving the same subnet.

In practice, only a few servers (perhaps one or two) would be configured to serve the same hosts, so any "storms" that did occur would likely be very mild or completely unnoticeable.

We at SRI run a script based on Van Jacobsen's *tcpdump* program that captures broadcast traffic, retaining the individual broadcast transactions for one hour. This record is very helpful in ascertaining the cause of a broadcast storm, as one can look at the past hour's broadcast packets and see which machines have participated in the storm and possibly which machine caused it.

Nervous Hosts

A "nervous host" is one that continually attempts to send packets to another host that is down. Since the first step in sending a packet to another host on an Ethernet is to broadcast an ARP packet, a large number of nervous hosts can result in an unusually large number of broadcast packets.

The classic "nervous host" is a UNIX host set up to print on a printer that is down and is connected directly to the network. Let us assume that the printer has been down for a long time, and that the host has not attempted recently to send a job to that printer.

Then let us also assume that a user queues a print job for the dead printer. As long as the printer is down, the host will repeat the following:

1. The printer daemon (*lpd*) will notice that there is a job to be printed and will initiate the filter process specified in the *printcap* entry for the printer.

2. The filter process will attempt to open a network connection to the printer; since the printer is down, this attempt will fail.

3. The filter process will notify the printer daemon of the failure. However, since the printer could come back up at any time, the filter process will indicate that this is a temporary failure. Therefore, the printer daemon will retry the print job.

Since the printer daemon must do several disk accesses in order to start printing a job, a very large number of nervous hosts would be necessary to affect the Ethernet significantly. However, the considerable volume of broadcast ARP packets can confuse monitoring programs (to say nothing of people!).

A good way to diagnose this problem is to look at a record of broadcast packets. If many of these packets are ARP requests for a printer, it is likely that the printer is down and that there are nervous hosts trying to access it.

The best way to solve this particular problem is to fix your printer, but applications should use *sleep(1)* where necessary to prevent this problem.

Load Imbalances

A load imbalance occurs on local-area networks that consist of several Ethernet segments connected by bridges or gateways when:

- One of the segments has much more traffic than the others, or
- Most of the traffic on one of the segments is not local to that segment. The configuration shown in Figure 2 allows only a single packet at a time to be transmitted between a work station and its server. Much better results may be obtained by running parallel Ethernet cables to split the load, as shown in Figure 3. This configuration allows up to two packets to be transmitted at a time, for example, file server A could transmit a packet to one of its work stations at the same time that file server B is transmitting a packet to one of its work stations.

In general, pairs of hosts that communicate with each other heavily should be placed on the same Ethernet segment. Running physically parallel Ethernet cables allows hosts to be moved easily from one cable to another, as required by changing patterns of usage.

Note that the load balancing problem may be alleviated by the advent of higher-speed networks, such as the 100 Megabit FDDI, although usage will almost certainly expand to fill the available bandwidth.

Summary

In short, to keep your network healthy, *rwhod* and *ipforwarding* must be disabled (even on BSD 4.3 systems, if they have more than one network interface) and the network load must be distributed with care.

The reader should keep in mind that the forgoing discussion is by no means exhaustive. There are many more interesting problems in the form of subnets, gateways, routing protocols, and other elements of network architecture and operations.

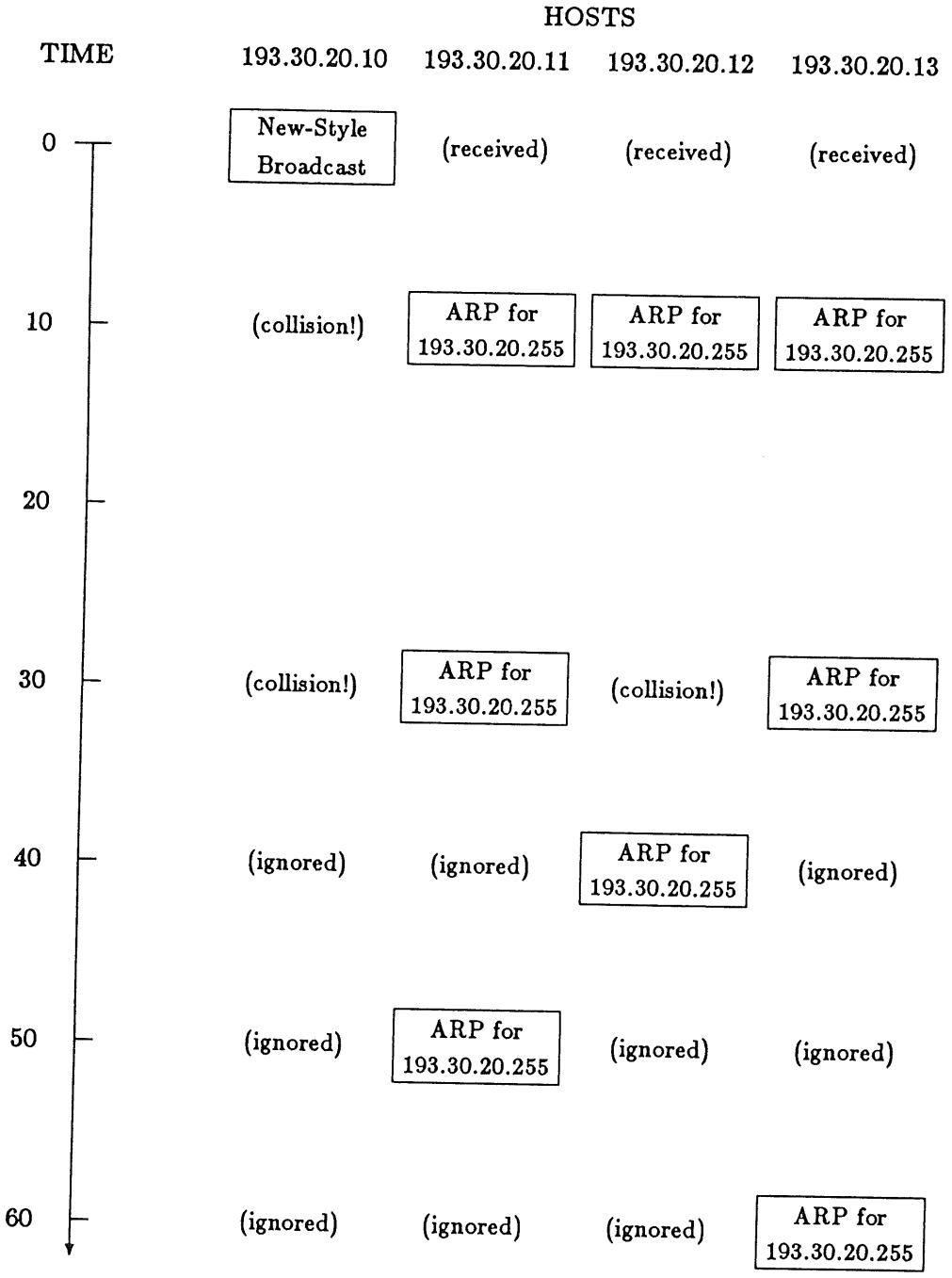


Figure 1: Classical Broadcast Storm

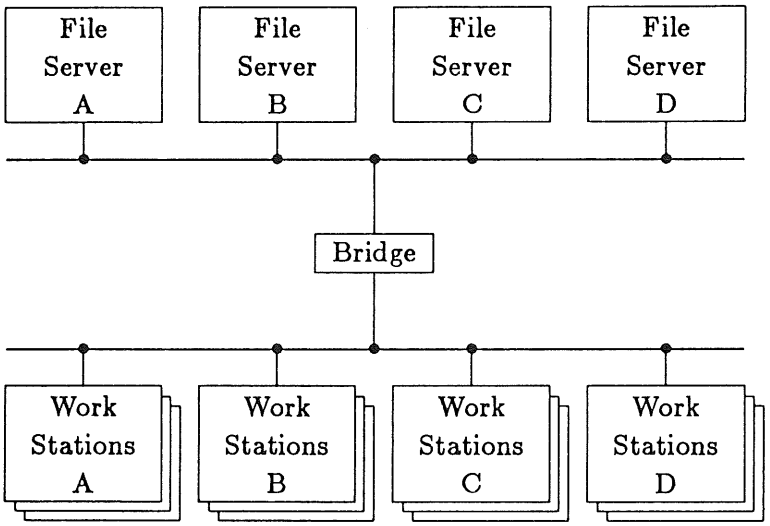


Figure 2: Poor Ethernet Configuration

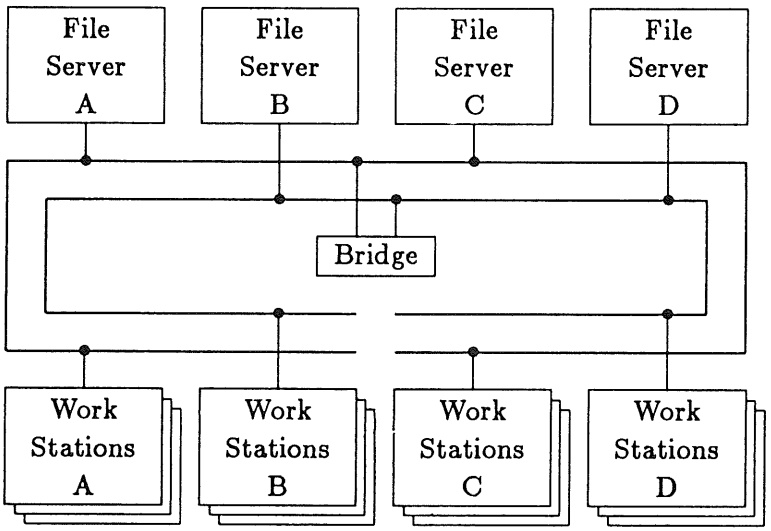


Figure 3: Better Ethernet Configuration

An Update on UNIX Standards Activities

Shane P. McCarron
NAPS International
August 1, 1988

This is the third in a series of reports on standards bodies relating to the UNIX community. Before I start, I would like to take a couple lines to thank all of those readers who were kind enough to drop me a line of either criticism or encouragement; both are greatly appreciated. In the future please feel free not only to comment on the articles here, but also on standards issues. I am more than happy to try and answer any of your questions either individually or through this column.

To business: the most important item to report (from my perspective) is that the USENIX Association has formed a Standards Watchdog Committee. The charter of this group is to keep an eye on as many of the standards efforts as possible, and report the progress of those efforts back to the membership. In addition, the group will be looking for important or contentious decisions, and trying to determine a USENIX position where it seems appropriate. The group will also be looking to you, the members, for input. Everyone has opinions, and the Watchdog Committee, through its standards committee representatives, can serve as a channel to get your ideas to the appropriate groups or can put you in contact with the appropriate people. For more information, please contact:

John S. Quarterman
Texas Internet Consulting
701 Brazos, Suite 500
Austin, TX 78701
(512) 320-9031
jsq@usenix.org, uunet!usenix!jsq

As always, the standards bodies have been pretty busy during the past quarter. Busy, that is, in standards body terms. There is often a great deal of heat, but very little light. I have remarked in the past that these committees can take a long time to complete things.

P1003.0 – The POSIX Open Systems Environment Guide

The IEEE 1003.0 working group met on July 12 & 13, 1988 in Denver, Colorado. The purpose of this meeting was to have the group members, who had volunteered during the March meeting to work on certain portions (sub-groups) of the POSIX Open Systems Environment guide document, present their material for review and critique by the group. This was accomplished on day 1 and the morning of day 2. The sub-groups that were discussed included:

1. Operating System
2. Database Management
3. Data Interchange
4. Network Services
5. User Interface
6. Languages
7. Graphics

The remainder of the meeting focused on goals and objectives for the next meeting in October. There was strong consensus within the group that the next goal should be a very rough draft document. Volunteers were assigned to each sub-group above with the purpose of putting into narrative form the material that had been presented. It was also agreed that distribution of this draft prior to the October meeting would be essential in order to allow for good, well thought-out discussion during the meeting.

The group has targeted October, 1989 as a goal for beginning the balloting process. This is aggressive, but possible, assuming that the effort between meetings can be maintained at its present level.

Overall, the meeting was very productive and is drawing more participation from a good cross-section of vendors and users.

P1003.1

The big news this month is, of course, that as of August 22nd the POSIX System Services Interface standard is complete. By the time you read this, final drafts should have been circulated to all of the POSIX working group members, and copies of that draft should be available from the IEEE office in New York. While you can obtain a copy of the final draft now, you would do well to wait for a couple of months and pick up a real, hard-bound version of the standard from the IEEE. To order a copy of the final draft, contact:

IEEE Standards Office
345 E. 47th St.
New York, NY 10017
(212) 705-7091

Since the last installment in this series, the 1003.1 standard has gone through not one, and not two, but three more recirculations. As you may remember, the second recirculation was scheduled to take place in May, and it did. This one went as well as expected, and generated some excellent feedback. The changes from that recirculation were assembled and sent back to the balloting group for review at the end of June. As a result of that recirculation, there were yet more changes to the standard, and those changes had to be recirculated as well. The final recirculation took place at the end of July, and generated no substantial changes. At that point the standard was released to the Technical Editor for final copy editing, and has now been balloted on and approved by the IEEE Standards Board!

This is actually good and bad. It is good for all the reasons you would suppose. It is bad because the standard is not perfect; there are things that shouldn't be in it that are (e.g. some weird timezone stuff and *read()* and *write()* functions that allow broken behavior), and things which should be in it but are not (like *seekdir()* and *telldir()*). Even though the standard is not perfect, at least we now have a foundation upon which additional documents can be based. In the future this standard will be extended and revised, but for now (in combination with Standard C), it's the best thing we have for application portability.

In the meantime, the .1 working group has not been idle. Although the initial work on the Services Interface standard was completed some months ago, there are always new areas to work in. The following is information on developments where they occurred.

Clean Up

There are some issues that were not handled to the satisfaction of the working group in the first cut of the standard. A small group is working on sifting through the unresolved balloting objections (there were several) and identifying those items that can be rectified through modification to the standard. It turns out that many of the unresolved objections were very reasonable items, but were introduced too late in the process to be placed in the standard. Those items will be looked at and possibly added to the standard in a supplement.

Language Independent Description

While little progress has been made in this area by the .1 working group, it turns out that there has been quite a bit of work done by other working groups and technical committees. The /usr/group technical committee on supercomputing, in particular, has produced a Fortran language description of the .1 interface. In the process they have come up with a number of items that can be used by the .1 people to develop their language independent description.

Terminal Interface Extensions

The Working Group looked at the various requirements necessary for a terminal interface standard (a terminal interface standard is something like the Terminal Interface Extensions in the SVID, better know as *curses/terminfo*). The group determined that there is little or no way to get a single interface standard that will satisfy the needs of the entire community. Those people with bit mapped displays can do things better, and we should let them. Those people with block mode terminals have special needs that should not have to be addressed by otherwise portable applications. The majority of users that we are trying to satisfy, those with character based

terminals, have well defined needs that are already being addressed by existing practice.

What's the solution? Well, none was really proposed, but I would guess that the people in the bit mapped world are going to care a lot more about Open Look and Presentation Manager (bite my tongue) than they are about something based on *terminfo* or *termcap*. For the other 90 percent of the UNIX-using community, while *terminfo* & *termcap* may be what they are used to seeing, it is hardly attractive enough to make them sit up and take notice. They are looking for flashier, better, faster applications, and the traditional interface is not going to be enough. For application developers, the functionality which can be achieved via *terminfo* is fine but hardly adequate for building the products that the user community is coming to expect.

I would guess that the POSIX committees will settle on some subset of the *terminfo* interface as the standard, and that no one will use it. Sure, it will be on every POSIX conformant system, but who cares? It is a lame interface, and someone will come up with a better one soon enough.

New Archive Format

As I mentioned previously, the ISO has asked P1003.1 to come up with a new archive format that will not have the deficiencies of *tar* or *cpio* and will be able to take the security concerns of the P1003.6 group into consideration (I assume that by this they mean access control lists, mandatory access controls, and the like). Little was done on this topic between meetings, but at the July meeting the committee discussed ways to extend the *cpio* archive format to take these things into consideration. While the technical details of this extension are clear, they are also boring. Suffice it to say that the filename field in the archive can be extended through a kludge and that it would be backward compatible.

This met with mixed reactions, and I believe that in the end it will not be used. This discussion (popularly known as Tar Wars) has been very religious and contentious, and I don't think that a format based on either will be able to get popular support from the working group. There is now a small group of people (from both camps) working on another new format, and I am certain that they will come up with something for the October meeting.

P1003.2 – Shell and Tools Interface

This group is actually a little bit ahead of schedule. Forget all the nasty things I have said about their schedule being too tight and optimistic – they are actually going to do it! You're not as impressed as I am, I can tell. Some people are just never satisfied. Okay, here's some evidence for you.

Functionality was frozen at the March meeting. This means that no additional commands or concepts could be added to the standard. It also means that the working group members were free to concentrate on the content of the draft, instead of looking at new proposals for additional commands all of the time. This has turned out to be very profitable; the draft has been cleaned up to the point where it can be submitted (to the working and corresponding groups) for a mock ballot in September. A mock ballot is just that – a process during which the draft is picked apart as it would be in the balloting process, with changes submitted through formal balloting objections. This may seem a little excessive, but it has proven effective in the past.

Assuming that all goes well, and the objections from the mock ballot are resolved at the October meeting, the group could go to a full ballot as early as January. A less optimistic scenario shows the group working on resolution of the mock ballot for two full meetings, with the real ballot occurring in February or March. Either way, the group is on schedule for a full use standard before the end of 1989.

In addition to this good news, there were a few key decisions made at the July meeting.

This side of the Tar Wars is apparently at an end. There were two aspects to the war – user-program interface and actual archive format. The interface side of it seems to have been settled by the introduction of a command called *pax* (Latin for peace). This command will be able to read and write both types of archives and has an interface that is acceptable to both camps. While this has not been agreed upon by the balloting group, or even by the full working group, the interface is pretty

familiar, and I believe it will be approved with little change.

The group also concentrated on trying to remove anything that might be considered implementation dependent from the draft. This included removing the octal modes from *chmod*, and the signal numbers from *trap* and *kill*. In their place go all of the mnemonic command line arguments that have been in those commands all along, but aren't used by anyone. As a committee member I can see what they are trying to do, and even agree with it. As a user, however, I wish they would have placed requirements that, say, `kill -9` would always map to SIGKILL. At least that way I wouldn't have to fix every shell script I have ever written.

P1003.3 – Testing and Verification

This working group is progressing well on its verification standard for 1003.1. They are planning to have a version to ballot in January or February of 1989. That would make the standard available just about the time that the major vendors are finishing their .1 conformant implementations.

The group has also started supplying liaison people to each of the other working groups. These people, with their experience writing a testing standard for .1, are proving very valuable in designing testable standards.

New POSIX Work Items

In addition to all of the committees you have heard about in past articles, there were several new working groups proposed to the P1003 steering committee.

System Administration

The committee recognizes the need for a standard interface to many of the system administration utilities that we are plagued with. While there was a considerable amount of skepticism exhibited from the members, the steering committee has agreed to let work progress on this topic. Consequently, a PAR was filed by Steve Carter of Bellcore, and the new working group will start meeting in October.

This group has a lot of work ahead of them; the difficulties of designing standard interfaces to things like *fsck* and *fsdb* may prove impossible. Also, from an system implementor standpoint, I would hate to have the administrative functions I can provide limited by something that a standards committee is going to generate based on existing practice. This is not an area in which there is a huge body of existing applications, so implementors should be allowed to innovate and improve if they like.

On the other hand, the computer users of the world are probably pretty sick of having to learn a new way to enable printers on every system they purchase. For those people, having a standard is going to be a big win. This is one of those times when the saying "be careful what you wish for..." comes into play. The ultimate, generic system administration interface may prove to be so restricted or brain-dead that it is of no use to anyone. The .1 standard was nearly that way.

Networking

Another new working group will be focusing on the services and service interfaces for a networked POSIX conformant system. While the exact charter and goals of this group are not fully established, what they are not trying to do is. They are not trying to overlap the work of the ISO-OSI committees, nor are they trying to supplant the work being done by IEEE 802. Their plan is to spend two years defining the services and service protocols, and maybe an additional year defining interfaces to those protocols.

User Interface Commands

If you have looked closely at the 1003.1 and .2 standards, you will notice that there is nothing in either of them about User Interface. Well, you're not alone, and someone is finally going to do something about it. A sub-group of the Shell and Tools committee has been formed to codify the interface of many of the classic UNIX commands (*vi*, *ed*, etc.). In addition, the group will be defining the user interface aspects of those commands already in the .2 standard which have traditionally had user interfaces as well as their programmatic ones.

This group is going to work somewhat in a vacuum – since there is no standard for

terminal interface, the user interfaces defined are not going to have a way, programmatically, of being put on the screen. *terminfo* will of course work for this, but it is not a standard. Hopefully the .1 committee can get a supplement out regarding this before the .2 subgroup finishes its work describing the utilities.

X/Open

The X/Open group is just about to release version 3 of the X/Open Portability Guide. This set of manuals is a must for any application developer or system implementor planning on marketing products in Europe. Version 3 will encompass all of the .1 standard, but will not contain any of the items proposed in the latest drafts of .2 – that document is too immature. The XPG also has language definitions, database interface specifications, and many other things that a growing programmer needs in the UNIX world.

NBS – Federal Information Processing Standard

I have written about this in each issue of this report, and each time I say that it is almost here. Well, I am done making predictions. The Federal government has a shield that my crystal ball just refuses to penetrate. I have heard recently that the FIPS for the .1 standard is within the Commerce department somewhere, but I have no proof. When it does finally come out, it will be based on a version of the standard that is almost a year out of date. Draft 12 of the .1 standard resembles the final standard about like a caterpillar resembles a butterfly. This is very unfortunate, as the vendors that are serious about selling computers to the Feds are going to have to conform to that standard, and not the real one. Note that while the NBS did try to jump the gun a little bit, they forced the .1 committee to work harder and faster. Without their encouragement the standard may well never have been finished.

Of course, the NBS has indicated that they will start making the FIPS conform to the final standard just as soon as it is out (now, that means). But, given the amount of time it took them to get the first standard out the door, I'm not holding my breath. It could be deep into 1989 before we see a revised FIPS hit the Federal Register's list of announcements.

In the meantime, the NBS is proceeding in its specification of other interim FIPS. Just until there are real standards in these areas, of course, we are going to see FIPS on Shell and Tools, User Interface, System Administration, Terminal Interface Extensions, and probably shoe lacing. The NBS people are very busy cranking out standards that Federal government departments can cite when generating bid requests. Unfortunately, in those cases where the committees aren't far enough along yet, these standards are going to be based on the SVID. And if the SVID is used as a base document by the Feds, you can be sure that it will also be used by any standards committees that come along later and want to "codify existing practice." Just another example of the Federal government guiding the standards community.

The NBS is putting on a series of workshops this fall to address some of these issues, and get input from the community. The first of these workshops, a seminar on "POSIX and other Application Portability Profile Standards" will be September 22nd and 23rd. For more information, contact Debbie Jackson at (301) 975-3295. She will be happy to send you registration materials, as well as give you information about future workshops being put on by the National Bureau of Standards.

X3J11 – ANSI C Language Standard

This standard is pretty important to everyone in the UNIX community. Unfortunately, that means that everyone has to get involved in the development of it, and that takes time. The document has now entered its third public comment period (July 1st → August 31st). From what I gather, the committee will be very reluctant (read "it will never happen") to make any substantive changes to the standard as a result of this period. What they are looking for is affirmation from the public that the changes made in round two were adequate to remove most of the outstanding objections.

The good news here is that the `noalias` keyword has been removed from the draft. This was a very contentious issue and was

introduced very late in the process. In simplest terms, `noalias` would allow the programmer to specify that the program, for that statement, would do exactly what it was supposed to do. Pretty silly, when you get right down to it. Anyway, its gone now – like a bad dream.

In addition, a number of simple editorial changes were made. Most of these are transparent, and just made the standard a little more readable. Unfortunately, it is still a standard written by programmers, for programmers, and is a little hard to read. There is even rumor of a *x3speak* program, like the *valspeak* of a few years ago, about to come out in *comp.sources.misc*. This would take any prose and render it senseless through the addition of legalese. My advice to future readers of the standard is this: Don't go into the water alone. Use the buddy system, and take a reader's guide with you.

Assuming all goes well at the September meeting, the ANSI C Language Standard should be published later this year.

Well, that's about it for this month. Remember, keep those cards and letters coming to:

Shane P. McCarron
NAPS International
117 Mackubin St., Suite 6
St. Paul, MN 55102

(612) 224-9239
ahby@bungia.mn.org

Future Events

EUUG Autumn Conference
Estoril, Portugal, Oct. 3-7

C++ Miniconference
Denver, CO, Oct. 17-21

The Program Chair is Andrew Koenig of AT&T. See page 3.

Large Installation
Systems Administration II
Monterey, CA, Nov. 17-18

The Program Chair is Alix Vasilatos of MIT's Project Athena. See page 6.

Japan UNIX Society
Osaka, Nov. 11-15, Conference & Exhibition
Toyko, Dec. 7-8, UNIX Fair '88

For both events, contact: Ms. Hiroko Tsunoda, Japan UNIX Society, 2-12-505 Hayabusa-cho, Chiyoda-ku, Tokyo 102, +81-3-234-5058

USENIX 1989 Winter Technical Conference
San Diego, Jan. 30-Feb. 3, 1989

See page 5.

EUUG Spring Conference
Brussels, Apr. 10-14, 1989

See page 7.

Long-term USENIX & EUUG Schedule

Jun 12-16 '89 Hyatt Regency, Baltimore
Sep 18-22 '89 Vienna, Austria
Jan 22-26 '90 Omni Shoreham, Washington, DC
Apr 23-27 '90 Munich, W. Germany
Jun 11-15 '90 Marriott Hotel, Anaheim
Jan 21-25 '91 Dallas
Jun 10-14 '91 Opryland, Nashville
Jan 20-24 '92 Hilton Square, San Francisco
Jun 8-12 '92 Marriott, San Antonio

Publications Available

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

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

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

We hope to have EUUG tapes and conference proceedings available shortly.

Conference and Workshop Proceedings

Meeting	Location	Date	Price	Overseas Mail	
				Air	Surface
USENIX	San Francisco	Summer '88	$20	$25	$5
C++ Workshop	Santa Fe	November '87	20	25	5
Graphics Workshop IV	Cambridge	October '87	10	15	5
USENIX	Wash. DC	Winter '87	10	25	5
Graphics Workshop III	Monterey	December '86	10	15	5

4.3BSD Manuals

The USENIX Association now offers all members of the Association the opportunity to purchase 4.3BSD manuals.†

The 4.3BSD manual sets are significantly different from the 4.2BSD edition. Changes include many additional documents, better quality of reproductions, as well as a new and extensive index. All manuals are printed in a photo-reduced 6"×9" format with individually colored and labeled plastic "GBC" bindings. All documents and manual pages have been freshly typeset and all manuals have "bleed tabs" and page headers and numbers to aid in the location of individual documents and manual sections.

A new Master Index has been created. It contains cross-references to all documents and manual pages contained within the other six volumes. The index was prepared with the aid of an "intelligent" automated indexing program from Thinking Machines Corp. along with considerable human intervention from Mark Seiden. Key words, phrases and concepts are referenced by abbreviated document name and page number.

While two of the manual sets contain three separate volumes, you may only order complete sets.

The costs shown below do not include applicable taxes or handling and shipping from the publisher in New Jersey, which will depend on the quantity ordered and the distance shipped. Those charges will be billed by the publisher (Howard Press).

Manuals are available now. To order, return a completed "4.3BSD Manual Reproduction Authorization and Order Form" to the USENIX office along with a check or purchase order for the cost of the manuals. You **must** be a USENIX Association member. Checks and purchase orders should be made out to "Howard Press." The manuals will be shipped to you directly by the publisher.

Manual	Cost*
User's Manual Set (3 volumes) User's Reference Manual User's Supplementary Documents Master Index	$25.00/set
Programmer's Manual Set (3 volumes) Programmer's Reference Manual Programmer's Supplementary Documents, Volume 1 Programmer's Supplementary Documents, Volume 2	$25.00/set
System Manager's Manual (1 volume)	$10.00

* Not including postage and handling or applicable taxes.

4.2BSD Manuals are No Longer Available

† Tom Ferrin of the University of California at San Francisco, a former member of the Board of Directors of the USENIX Association, has overseen the production of the 4.2 and 4.3BSD manuals.

Local User Groups

The USENIX Association will support local user groups by doing an initial mailing to assist the formation of a new group and publishing information on local groups in *;login:*. At least one member of the group must be a current member of the Association.

CA - Fresno: the Central California UNIX Users Group consists of a *uucp*-based electronic mailing list to which members may post questions or information. For connection information:

Educational and governmental institutions:

Brent Auernheimer (209) 294-4373
brent@CSUFresno.edu or csufres!brent

Commercial institutions or individuals:

Gordon Crumal (209) 875-8755
csufres!gordon (209) 298-8393

CA - Los Angeles: the Los Angeles UNIX Group meets on the 3rd Thursday of each month in Redondo Beach.

Drew Bullard (213) 535-1980
{ucbvax,ihnp4}!trwrb!bullard

Marc Ries (213) 535-1980
{decvax,sdcrdcf}!trwrb!ries

CO - Boulder: meets monthly at different sites.

Front Range UNIX Users Group
USENIX Association Exhibit Office
5398 Manhattan Circle
Boulder, CO 80303

John L. Donnelly (303) 499-2600
{boulder,usenix}!johnd

FL - Coral Springs:

S. Shaw McQuinn (305) 344-8686
8557 W. Sample Road
Coral Springs, FL 33065

FL - Melbourne: the Space Coast UNIX Users Group meets at 8pm on the 3rd Wednesday of each month at the Florida Institute of Technology.

Alex Stover (305) 724-3962
codas!lola!als

Bill Davis (305) 242-4449
bill@ccd.harris.com

FL - Orlando: the Central Florida UNIX Users Group meets the 3rd Thursday of each month.

Mike Geldner (305) 862-0949
codas!sunfla!mike

Ben Goldfarb (305) 275-2790
goldfarb@hcx9.ucf.edu

Mikel Manitius (305) 869-2462
{codas,attmail}!mikel

FL - Tampa Bay: the Tampa UNIX Users Group meets the 1st Thursday of each month, alternately in Largo and Tampa.

Scott Stone (813) 974-3307
uflorida!usfvax2!stone, stone@usf.edu

Bill Hargen (813) 530-8655
{codas,usfvax2}!pdn!hargen

George W. Leach (813) 530-2376
uunet!pdn!reggie

GA - Atlanta: meets on the 1st Monday of each month in White Hall, Emory University.

Atlanta UNIX Users Group
P.O. Box 12241
Atlanta, GA 30355-2241

Marc Merlin (404) 442-4772
Mark Landry (404) 365-8108

MI - Detroit/Ann Arbor: meets the 2nd Thursday of each month in Ann Arbor.

William Bulley (313) 995-6211
web@applga.uucp

Rich McGill (313) 971-5950
rich@oxtrap.uucp

Steve Simmons (313) 426-8981
scs@lokkur.uucp

MI - Detroit/Ann Arbor: dinner meetings the 1st Wednesday of each month.

Linda Mason (313) 855-4220
michigan!/usr/group
P.O. Box 189602
Farmington Hills, MI 48018-9602

MN - Minnetonka: meets the 1st Wednesday of each month.

UNIX Users of Minnesota
545 Ashland Avenue #3
St. Paul, MN 55102

Scott Anderson (612) 688-0089
scott@questar.mn.org

MO - St. Louis:

St. Louis UNIX Users Group
Plus Five Computer Services
765 Westwood, 10A
Clayton, MO 63105

Eric Kiebler (314) 725-9492
ihnp4!plus5!sluug

NE - Omaha: meets on the 2nd Thursday of each month.

/usr/group nebraska
P.O. Box 44112
Omaha, NE 68144

Sukan Makmuri (402) 422-8367
ihnp4!ugn!root

New England - Northern: meets monthly at different sites.

Emily Bryant (603) 646-2999
Kiewit Computation Center
Dartmouth College
Hanover, NH 03755

David Marston (603) 883-3556
Daniel Webster College
University Drive
Nashua, NH 03063

decvax!dartvax!nneuug-contact

NJ - Princeton: the Princeton UNIX Users Group meets monthly.

Pat Parseghian (609) 452-6261
Dept. of Computer Science
Princeton University
Princeton, NJ 08544

pep@Princeton.EDU

NY - New York City:

Unigroup of New York
G.P.O. Box 1931
New York, NY 10116

Ed Taylor (212) 513-7777
{attunix,philabs}!pencom!taylor

New Zealand:

New Zealand UNIX Systems User Group
P.O. Box 13056
University of Waikato
Hamilton, New Zealand

OK - Tulsa:

Pete Rourke
$USR
7340 East 25th Place
Tulsa, OK 74129

PA - Philadelphia: the UNIX SIG of the Philadelphia Area Computer Society (PACS) meets the morning of the 3rd Saturday of each month at the Holroyd Science Building, LaSalle University.

G. Baun, UNIX SIG
c/o PACS
Box 312
La Salle University
Philadelphia, PA 19141

{inhp4,cbosgd,rutgers}!{bpa,cbmvax}!
temvax!pacsbb!{gbaun,whutchi}

TX - Dallas/Fort Worth:

Dallas/Fort Worth UNIX Users Group
Seny Systems, Inc.
5327 N. Central, #320
Dallas, TX 75205

Jim Hummel (214) 522-2324

TX - San Antonio: the San Antonio UNIX Users (SATUU) meets the 3rd Wednesday of each month.

Jeff Mason (512) 494-9336
Hewlett Packard
14100 San Pedro
San Antonio, TX 78232

gatech!petro!hpsatb!jeff

WA - Seattle: meets monthly.

Bill Campbell (206) 232-4164
Seattle UNIX Group Membership Information
6641 East Mercer Way
Mercer Island, WA 98040

uw-beaver!tikal!camco!bill

Washington, D.C.: meets the 1st Tuesday of each month.

Washington Area UNIX Users Group
2070 Chain Bridge Road, Suite 333
Vienna, VA 22180

Samuel Samalin (703) 448-1908

Minutes of the AUUG Management Committee Meeting
May 13, 1988

1. The meeting opened at 10:06. Present were Chris Campbell (CC) (arrived late, as noted below), Piers Dick-Lauder (PL), Robert Elz (KRE), John Lions (JL) in the chair, Chris Maltby (CM), Tim Roper (TR), and Peter Wishart (PW). Also present was John Carey (JC), the AUUGN editor. John O'Brien (JO'B), the returning officer, attended part of the meeting, as noted.
2. JL opened the meeting by welcoming PW to his first meeting as a member of the AUUG executive. PW apologised for being unable to attend the previous meeting.
3. The minutes of the previous meeting (February 1988), which had been circulated earlier, were tabled.
4. TR pointed out that the second sentence of point 4, item 32, should be deleted.
5. Moved (CM, seconded TR) **That the minutes, as amended be accepted.** Carried (5/0 PW abstaining).
6. Business arising from the minutes

   Items carried forward from previous meetings

   — **NSWIT meeting new members:** More complaints have been received. No solution appears imminent.
   — **Meeting Guidelines Documents:** PL distributed the results of his work. He wants feedback, and has an electronic copy available.
   — **Usenix Journal (Computing Systems):** The secretary has been in communications with Usenix, and the initial issue should be on its way. Cost for a subscription for AUUG members will be $US23. It was suggested that an application form for subscriptions should be placed in the June AUUGN.

   Item 9 **Newsletter:** JC indicated that the publication date appears on the cover, as requested, starting from volume 9, issue 2.

   Item 11 **Newsletter sales tax:** The solicitors have indicated that we do have to pay this. A letter containing their detailed advice is on its way, but has not yet been received.

   Item 13 **AUUG database location:** The database has been moved to Melbourne.

   Item 20 **September AUUG meeting guests:** Mashey has now agreed to attend.

Item 27 **Database, Post Office Box, etc:** The database has been moved, the post office box has not yet been redirected. No secretarial assistance has been obtained yet.

Item 34 **Conference advertising:** Nothing done yet.

Item 35 **Thompson publicity:** No specific action as regards AUUG, however it was noted that Thompson has been interviewed on ABC radio.

Item 38 **NSWIT conference returns:** Nothing heard from Greg Webb recently, however PL provided evidence that Webb is still alive, having had mail on other matters. The president and treasurer are to visit him, and attempt to obtain a cheque, and all relevant documentation, especially registration & membership forms.

Item 39 **Meeting CFP:** Done.

Item 40 **Badges for meeting attendees:** Done. The badges are available, and an invoice has been received.

10:27: At this point (coincidentally) CC arrived bearing the badges with him, and apologising for his late arrival. Samples of the badges were shown to the committee, which resulted in general approval. CC stated that he was not entirely pleased with the result, and that he was negotiating with the manufacturer, with the aim of having the price reduced.

TR took charge of the badges.

Item 45 **ACMS meeting responsibilities:** The secretary has been in contact with ACMS, a full report to follow later.

Item 47 **AUUG conference organiser job description:** No information on progress. Last word from Greg Webb was that it would be done in a week, two weeks ago.

Item 53 **Solicitor & Trade Marks:** The solicitor has been contacted about this, without any result so far.

Item 54 **Change of name:** Ballot was not held as intended, current plan is to combine it with the AUUG election ballot.

Item 60 **Accountant to prepare books:** Not done yet.

Item 62 **Call for nominations:** Done. JO'B will attend this meeting later, and disclose the outcome.

7. JL gave the president's report. He indicated that he had attended the EUUG meeting in London, as a representative of AUUG. He agreed to write a report of this meeting for AUUGN. He had also given a talk at a monthly SINIX

(Singapore user group) meeting.

8. Moved (CM, seconded CC) **That the president's report be accepted.** Carried (7/0).

9. The secretary presented his report. He distributed a summary of the membership status, and indicated that it showed a drop in membership numbers. Some correspondence with ACMS was tabled, setting out the precise role that ACMS will play in the meeting organisation.

10. The secretary also indicated that he had attended a meeting of Pacific area user groups, in Singapore in April. The groups agreed to exchange newsletters, with English summaries (but not translations) where appropriate. A Pacific area users meeting is planned for Singapore sometime in 1989.

11. The database has been transferred to Melbourne, and the secretary has created perl scripts (using Larry Wall's public domain *Perl* programming language) to manipulate it.

12. Moved (TR, seconded PL) **That the secretary's report be accepted.** Carried (7/0).

13. JL amended the president's report. He indicated that AUUG was represented by himself, and Greg Rose at Ross Nealon's funeral, and had presented flowers in the name of AUUG.

14. Moved (JL, seconded TR) **That AUUG make a donation of $200 to the oncology unit at Wollongong Hospital, in Ross Nealon's name.** Carried (7/0).

15. JL will prepare an obituary notice for AUUGN, to accompany one from Juris Reinfelds. JL indicated that Richard Miller may send one as well.

16. The treasurer presented his report. He distributed copies of the balance sheet, and indicated that he had still received no funds from NSWIT after the September 87 AUUG meeting, and nor had any additional funds been moved to a higher interest account.

17. The treasurer indicated that redirection of the post office box has not yet been done, and that the accounts have not yet been audited.

18. Moved (PL, seconded CC) **That the treasurer be instructed to have an audit carried out before the new committee takes over, with AUUG to bear the costs.**

19. Amendment (moved JL) **That PL be responsible for finding an auditor** was not seconded.

20. Amendment (moved TR, seconded CC) **That "before the new committee takes over" be altered to "before the end of May 1988".** Carried (7/0).

21. Motion, as amended, carried (7/0).

22. Moved (CC, seconded PW) **That the treasurer's report be accepted** Carried (7/0).

23. The newsletter editor (JC) presented a report. He indicated that he had changed jobs, and was currently putting together production facilities at his new employer's, Webster Computer Corporation. He indicated that Monash University Computer Science had agreed to act as a backup production facility.

24. JC indicated that he was having isolated problems with the printers, who need continuous chasing to get things done. The April issue of AUUGN is currently in the printing/distribution system somewhere.

25. JC is to talk to the postmaster about the occasional distribution problems that have been reported.

26. He also indicated that he still wants to rid himself of the store of old AUUGN copies that he currently has.

27. JC also indicated that AUUGN would achieve printing economies with either more or less members. He also indicated that AUUG continually needs more content.

28. Moved (PL, seconded CC) **That the editor's report be accepted.** Carried (7/0).

29. CC presented a report of the last ACSnet SIG meeting, held Tuesday May 10. He indicated that another meeting was to be held in the first or second week of June, after Bob Kummerfeld has the backbone information. Statistics that have been collected need to be analysed, an approach is to be made to Ron Baxter for assistance. Various methods of assistance, etc, are still being discussed.

30. PL indicated that an offer from NASA for assistance with a link to the US has been made. Also DSTO, Salisbury, SA, are contemplating US connections.

31. KRE indicated that the solicitors need more information on the aims of any company to be established, before giving any advice on the matter.

32. Incorporation: There has been no change since the last meeting, everything remains on hold until the result of the change of name ballot is known.

33. TR indicated that he had searched the yellow pages (Melbourne) to see if any secretarial assistance was available. He had one hit, which had looked promising, but was probably going to fail (the hit was on a moving target). He was not yet sure of the outcome.

34. TR agreed to contact Sigma Data about possible secretarial assistance. They have already contacted about a possible upgrade to the **auug.oz** host, and had been responsive.

35. TR distributed a list of duties required for secretarial assistance that he had prepared. TR is continuing his search.

36. No budget was available.

37. JO'B arrived at 11:57, and presented the committee with the list of nominees received for positions on the committee at the coming election.

38. Moved (CM, seconded PW) **That a referendum be sent to members with the election papers, "That the name of the group be altered from 'Australian Unix systems User Group Incorporated' to 'AUUG Incorporated'".**

39. JL and TR departed before the motion was put to a vote, CC was elected to take the chair.

40. Motion carried (5/0).

41. The meeting adjourned for lunch at 12:15, and resumed at 14:25 without JO'B, who was preparing the ballot papers.

42. There was much discussion of the September AUUG meeting. A conference organiser is needed. Matter still remaining to be attended to, include advertising, a mailout, and brochure preparation.

43. TR is to liaise with Wael Foda at ACMS, and see how much he is prepared to do. PW agreed to act as overall co-ordinator. CC agreed to handle press releases, and advertisements.

44. It was agreed that the deadline for early registration be altered from June 30 to July 15.

45. The meeting adjourned at 15:16, and resumed at 15:40.

46. A brief discussion on benefits for institutional members took place. Subscriptions to Computing Systems are to be arranged. A possible 20% reduction in AUUGN advertising rates was mentioned. Discussion was deferred to the next meeting.

47. Discussion of possible constitution changes was deferred to the next meeting. KRE to prepare a list of possible changes.

48. The next committee meeting is to be held on September 12, at Melbourne University, commencing at 13:00. KRE to arrange a room.

49. Some discussion of the possibilities of ties with the SUN local user group (and other similar groups) took place. Assuming that the members of that group desired it, would AUUG treat them as a chapter. No decision was taken.

50. The meeting closed at 16:17, after which members of the committee (and JO'B) commenced stuffing envelopes for the election and referendum ballot. The

secretary took the stuffed envelopes with him, to affix labels, and arrange postage as soon as possible.

THIS PAGE INTENTIONALLY LEFT BLANK

AUUG

Membership Categories

Once again a reminder for all "members" of AUUG to check that you are, in fact, a member, and that you still will be for the next two months.

There are 4 membership types, plus a newsletter subscription, any of which might be just right for you.

The membership categories are:

Institutional Member
Ordinary Member
Student Member
Honorary Life Member

Institutional memberships are primarily intended for university departments, companies, etc. This is a voting membership (one vote), which receives two copies of the newsletter. Institutional members can also delegate 2 representatives to attend AUUG meetings at members rates. AUUG is also keeping track of the licence status of institutional members. If, at some future date, we are able to offer a software tape distribution service, this would be available only to institutional members, whose relevant licences can be verified.

If your institution is not an institutional member, isn't it about time it became one?

Ordinary memberships are for individuals. This is also a voting membership (one vote), which receives a single copy of the newsletter. A primary difference from Institutional Membership is that the benefits of Ordinary Membership apply to the named member only. That is, only the member can obtain discounts on attendance at AUUG meetings, etc, sending a representative isn't permitted.

Are you an AUUG member?

Student Memberships are for full time students at recognised academic institutions. This is a non voting membership which receives a single copy of the newsletter. Otherwise the benefits are as for Ordinary Members.

Honorary Life Memberships are a category that isn't relevant yet. This membership you can't apply for, you must be elected to it. What's more, you must have been a member for at least 5 years before being elected. Since AUUG is only just approaching 3 years old, there is no-one eligible for this membership category yet.

Its also possible to subscribe to the newsletter without being an AUUG member. This saves you nothing financially, that is, the subscription price is the same as the membership dues. However, it might be appropriate for libraries, etc, which simply want copies of AUUGN to help fill their shelves, and have no actual interest in the

contents, or the association.

Subscriptions are also available to members who have a need for more copies of AUUGN than their membership provides.

To find out if you are currently really an AUUG member, examine the mailing label of this AUUGN. In the lower right corner you will find information about your current membership status. The first letter is your membership type code, N for regular members, S for students, and I for institutions. Then follows your membership expiration date, in the format exp=MM/YY. The remaining information is for internal use.

Check that your membership isn't about to expire (or worse, hasn't expired already). Ask your colleagues if they received this issue of AUUGN, tell them that if not, it probably means that their membership has lapsed, or perhaps, they were never a member at all! Feel free to copy the membership forms, give one to everyone that you know.

If you want to join AUUG, or renew your membership, you will find forms in this issue of AUUGN. Send the appropriate form (with remittance) to the address indicated on it, and your membership will (re-)commence.

As a service to members, AUUG has arranged to accept payments via credit card. You can use your Bankcard (within Australia only), or your Mastercard by simply completing the authorisation on the application form.

AUUG

Application for Ordinary, or Student, Membership Australian UNIX* systems Users' Group.

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

To apply for membership of the AUUG, complete this form, and return it with payment in Australian Dollars, or credit card authorisation, to:

AUUG Membership Secretary
P O Box 366
Kensington NSW 2033
Australia

- Please don't send purchase orders — perhaps your purchasing department will consider this form to be an invoice.
- Foreign applicants please send a bank draft drawn on an Australian bank, or credit card authorisation, and remember to select either surface or air mail.

I, ... do hereby apply for

☐ Renewal/New* Membership of the AUUG	$65.00	
☐ Renewal/New* Student Membership	$40.00	(note certification on other side)
☐ International Surface Mail	$10.00	
☐ International Air Mail	$50.00	
Total remitted	AUD$_____	(cheque, money order, credit card)

* Delete one.

I agree that this membership will be subject to the rules and by-laws of the AUUG as in force from time to time, and that this membership will run for 12 consecutive months commencing on the first day of the month following that during which this application is processed.

Date: __/__/__ Signed: _____

☐ Tick this box if you wish your name & address withheld from mailing lists made available to vendors.

For our mailing database - please type or print clearly:

Name: ..... Phone: ..... (bh)

Address: .... (ah)

..... Net Address: .....

.....

..... *Write "Unchanged" if details have not*

..... *altered and this is a renewal.*

Please charge $_____ to my ☐ Bankcard ☐ Visa ☐ Mastercard.

Account number: _____ _____ _____ _____. Expiry date: __/__.

Name on card: _____ Signed: _____

Office use only:

Chq: bank _____ *bsb* ____-____ *a/c* _____ *#* _____ ___

Date: __/__/__ *$* *CC type* ___ *V#* _____

Who: _____ *Member#* _____

Student Member Certification *(to be completed by a member of the academic staff)*

I, ... certify that

..... *(name)*

is a full time student at ..... *(institution)*

and is expected to graduate approximately ___/___/___.

Title: _____ Signature: _____

AUUG

Application for Institutional Membership
Australian UNIX* systems Users' Group.

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

To apply for institutional membership of the AUUG, complete this form, and return it with payment in Australian Dollars, or credit card authorisation, to:

AUUG Membership Secretary
P O Box 366
Kensington NSW 2033
Australia

• Foreign applicants please send a bank draft drawn on an Australian bank, or credit card authorisation, and remember to select either surface or air mail.

..... does hereby apply for

☐ New/Renewal* Institutional Membership of AUUG	$300.00
☐ International Surface Mail	$ 20.00
☐ International Air Mail	$100.00
Total remitted	AUD$_____

(cheque, money order, credit card)

* Delete one.

I/We agree that this membership will be subject to the rules and by-laws of the AUUG as in force from time to time, and that this membership will run for 12 consecutive months commencing on the first day of the month following that during which this application is processed.

I/We understand that I/we will receive two copies of the AUUG newsletter, and may send two representatives to AUUG sponsored events at member rates, though I/we will have only one vote in AUUG elections, and other ballots as required.

Date: __/__/__ Signed: _____

Title: _____

☐ Tick this box if you wish your name & address withheld from mailing lists made available to vendors.

For our mailing database - please type or print clearly:

Administrative contact, and formal representative:

Name: ..... Phone: ..... (bh)

Address: .... (ah)

..... Net Address: .....

.....

.....

.....

Write "Unchanged" if details have not altered and this is a renewal.

Please charge $_____ to my/our ☐ Bankcard ☐ Visa ☐ Mastercard.

Account number: _____ _____ _____ _____. Expiry date: __/__.

Name on card: _____ Signed: _____

Office use only: **Please complete the other side.**

Chq: bank _____ *bsb* _____-_____ *a/c* _____ *#* _____

Date: __/__/__ *$* *CC type* ___ *V#* _____

Who: _____ *Member#* _____

Please send newsletters to the following addresses:

Name: ..... Phone: ..... (bh)
Address: .... (ah)
..... Net Address: .....
.....
.....
.....

Name: ..... Phone: ..... (bh)
Address: .... (ah)
..... Net Address: .....
.....
.....
.....

Write "unchanged" if this is a renewal, and details are not to be altered.

Please indicate which Unix licences you hold, and include copies of the title and signature pages of each, if these have not been sent previously.

Note: Recent licences usally revoke earlier ones, please indicate only licences which are current, and indicate any which have been revoked since your last membership form was submitted.

Note: Most binary licensees will have a System III or System V (of one variant or another) binary licence, even if the system supplied by your vendor is based upon V7 or 4BSD. There is no such thing as a BSD binary licence, and V7 binary licences were very rare, and expensive.

☐ System V.3 source
☐ System V.2 source
☐ System V source
☐ System III source
☐ 4.2 or 4.3 BSD source
☐ 4.1 BSD source
☐ V7 source
☐ Other *(Indicate which)* .....

☐ System V.3 binary
☐ System V.2 binary
☐ System V binary
☐ System III binary

AUUG

Application for Newsletter Subscription
Australian UNIX* systems Users' Group.

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

Non members who wish to apply for a subscription to the Australian UNIX systems User Group Newsletter, or members who desire additional subscriptions, should complete this form and return it to:

AUUG Membership Secretary
P O Box 366
Kensington NSW 2033
Australia

- Please don't send purchase orders — perhaps your purchasing department will consider this form to be an invoice.
- Foreign applicants please send a bank draft drawn on an Australian bank, or credit card authorisation, and remember to select either surface or air mail.
- Use multiple copies of this form if copies of AUUGN are to be dispatched to differing addresses.

Please *enter / renew* my subscription for the Australian UNIX systems User Group Newsletter, as follows:

Name: ..... Phone: ..... (bh)

Address: .... (ah)

..... Net Address: .....

.....

..... *Write "Unchanged" if address has*

..... *not altered and this is a renewal.*

For each copy requested, I enclose:

☐ Subscription to AUUGN	$ 65.00
☐ International Surface Mail	$ 10.00
☐ International Air Mail	$ 50.00

Copies requested (to above address) _____

Total remitted AUD$_____

(cheque, money order, credit card)

☐ Tick this box if you wish your name & address withheld from mailing lists made available to vendors.

Please charge $_____ to my ☐ Bankcard ☐ Visa ☐ Mastercard.

Account number: _____ _____ _____ _____. Expiry date: __/__.

Name on card: _____ Signed: _____

Office use only:

Chq: bank _____ *bsb* ____-____ *a/c* _____ *#* _____

Date: __/__/__ *$* *CC type* ___ *V#* _____

Who: _____ *Subscr#* _____

AUUG

Notification of Change of Address
Australian UNIX* systems Users' Group.

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

If you have changed your mailing address, please complete this form, and return it to:

AUUG Membership Secretary
P O Box 366
Kensington NSW 2033
Australia

Please allow at least 4 weeks for the change of address to take effect.

Old address (or attach a mailing label)

Name: ..... Phone: ..... (bh)

Address: .... (ah)

..... Net Address: .....

.....

.....

.....

New address (leave unaltered details blank)

Name: ..... Phone: ..... (bh)

Address: .... (ah)

..... Net Address: .....

.....

.....

.....

Office use only:

Date: ___/___/___

Who: _____ *Memb#* _____