NEWSLETTER

EUUG

European UNIX† systems User Group

Volume 6, No. 2
SPRING 1986

CONTENTS

EUUG

European UNIX† Systems User Group

Newsletter Vol 6 No 2

Spring 1986

† UNIX is a Trademark of Bell Laboratories.

Jean Wood

EUUGN Editor

Want to know how to NOT win friends and influence people? It's easy. Become the editor of a newsletter. Some of your former friends and colleagues will have good wishes for you; some will have suggestions for the kinds of articles that should be in the newsletter; others will critique your existing work. Nearly none of these former friends and colleagues will offer you what you desparately need most: things to publish in the newsletter.

> Make your newletter editor happy today.
> Submit something for the European Unix systems User Group Newsletter.

Last time, I talked about my plans for this Newsletter; more of this and less of that, resulting in something of more value than we had before. I have learned that, unfortunately, one can only be selective if one has things from which to select.

> Make your newletter editor happy today.
> Submit something for the European Unix systems User Group Newsletter.

The European UNIX systems User Group is an affiliation of eleven national groups. In each and every governing board meeting, representatives from these eleven groups are urged to encourage contributions from their organisations and to write articles about their general activites. Last issue, we heard from the Netherlands. This time the Finnish and the Danish groups have contributed. What success, we've doubled the 'news' from national groups. Can we continue this trend?

> Make your newletter editor happy today.
> Submit something for the European Unix systems User Group Newsletter.

A number of people have recently made me VERY HAPPY. They submitted something original for publication in this issue. Many thanks to

> Peter Collinson
> Michel Gien
> Johan Helsingius
> Frank Kuiper
> Keld J:orn Simonsen

Hmmmm. It seems that more than half of these wonderful people are members of the EUUG Executive Board. Unfortunately for them, they are forced to listen to me rant and rave, bitch and moan, and beg and plead for contributions even more than the rest of the membership. Is this the newsletter of the European Unix systems Users Group or it's Executive Board?

> Make a Newletter Editor happy today.
> Submit something for the European Unix systems User Group Newsletter.

Serious technical papers, opinionated essays, original cartoons, suggestions and critisizms can be sent electronically to mcvax!euug-n (note small change) or mailed to me,

> Jean Wood
> Editor, EUUGN
> Digital Equipment
> Centre Technique (European)
> B.P. 29 Sophia—Antipolis
> 06562 Valbonne FRANCE

The deadline for submissions for the Autumn issue is 1 July 1986. The deadline for submissions for the Winter issue is 1 October 1986.

A happy newsletter editor produces a better newsletter. Then, everyone benefits.

Jean Wood

Editorial or 'A letter from Canterbury'

Peter Collinson

Secretary — EUUG

Our new editor, Jean Wood rang me up just before Easter and asked me to generate an editorial before Florence. So, here I am, the Sunday before the Florence Conference creating this document not on UNIX using some editor which we all know and love but on an Epson PX-8 laptop machine using Wordstar. I am not sitting on a chair in a terminal room talking via RS232 to a computer being shared by several other users but lying in bed half listening to the 'Archers' on the radio.‡ I use all this seemingly irrelevant stuff to show how the face of computing has changed since I first got involved with UNIX around 10 years ago.

I only got interested by accident. My boss (always with an ear to the ground and an eye for things that will catch on) had heard about it from Queen Mary College and we had managed to get the license and the RKO5's which were to run in the PDP-11/40. Steve Binns had booted the system but then had to go off on a familiarisation course for our brand new ICL-2960 and so I took over. From such beginnings....

It has always been hard to explain to people who came later quite why that system captivated so many people and laid the groundwork for UNIX as we know it today. One reason, I think, is isolation. We had the sources of everything and the temptation to tailor things from how they *did* work into how they *should* work was overwhelming. There was a snag to this — if you broke something then you had to mend it. But there was no software support centre, no network to emit panic messages on, no friendly guru to phone to solve the problem, and in my case nobody else on the site who had a faint understanding of the problems so that I could discuss their solution.

There was just the uncompromising, largely uncommented, often enigmatic source and you just had to get down a read a lot of C. As a result, you began to get a good understanding about how everything worked. However, there were the frustrating times too — when a phone call or a discussion would have saved many weeks banging away at some bug or other.

It is faintly possible that this year sees the 10th anniversary of the birth of the UK UNIX User Group. It was this meeting which began to break down the isolation. The User Group grew into the EUUG and has been responsible for putting people in touch with each other. To me, this has been its most important function. It worries me that some National groups are now looking at the organisation as it is today and saying 'what can we get out of it?' The answer is probably nothing very tangible; but the group has destroyed the isolation of the early days and this was done by people meeting people.

Without the User Group there would probably be no coherent UUCP network in Europe. It would have perhaps grown in little national networks and would not be seen as the important mechanism of communication which we see today. This is another way of banishing isolation; in a world of electronic mail, how do you meet people? How do you find out who is doing what? How do you obtain help for curing a particular software problem or opinion on hardware? Today, it is simple. You just broadcast a request in the news to several thousand machines world-wide and hopefully get back some answer — or at least perhaps an idea of where to look. It is now possible to have long and involved conversations in the mail with people on the other side of the world who you will perhaps never meet.

So, thankfully, things will never be quite the same again. Of course, you will say: UNIX has changed, we are not hacking kernels now; we have books on UNIX, we have binary systems, we have support from software houses, we have standardisation, we don't need gurus now. Well, actually, perhaps the real problem is the UNIX *hasn't* changed much but the people using it have; and perhaps more importantly, the people selling it have. It seems to me that EUUG can and should provide an unbiased forum for discussion and information which will hopefully dispel salesman's hype and show things for what they really are.

‡ A 'soap' opera whose charm is about as indescribable (and some would say inexecrable) as that of UNIX.

WHY the EUUG Autumn'86 MANCHESTER WORKSHOP ?

Michel Gien

Manchester Workshop Program Chairman

For the past few years, EUUG conferences have shown a continuous growth: more and more people attend, exhibitions are bigger and bigger, and venues become more fancy. On the other hand, financial risks are higher, accomodations are more expensive and communications, technical discussions and exchange between participants, which use to be one of the main interest of these UNIX get-together, tend to be more difficult. As the size and importance of the event increases, it tends to become more "commercially" oriented. Technical newcomers to the UNIX community and students may eventually have problems to even justify attending these events, if they can afford it !

The EUUG decided to experiment with the idea of renewing with oldies but goldies conferences, for one of the two annual EUUG events, and try to keep one of them "beautifully small".

The "new style" intended for EUUG future conferences is therefore to hold one big, noisy, lively, and very interesting conference associated with a commercial exhibition, and one small, quiet, lively, and very interesting workshop (note the change in the appellation), with an emphasis on technical discussions.

The Manchester Autumn'86 Workshop is a move towards this new (old) direction.

To keep it small, instead of arbitrarily reducing the number of attendees, it was decided to limit the subject to *one* theme. But, to keep it sufficiently attractive, the theme has been chosen among those where most energy is being spent these days in the UNIX technical community : Distributed systems and applications.

To keep it small, the location has been chosen in a grey (green ? red ?), rainy, industrial city, in the middle of nowhere (do they call it England ? :-). But to keep it attractive and "communicative", it will be cheap, accomodations will be of the student "friendly" type, and the conference beer will be EUUG "appelation controle'e".

Distributed UNIX Systems and Applications

Limiting to this theme is intended for:

—Focusing people to those interested by the subject,

—Focusing technical presentations and discussions in order to dive deeper into the real matter,

—Covering *all* technical aspects related to *the* theme.

Focusing people does not mean that it should be limited to the research community. It just means that technical people, actually working on the subject, or with deep interests for it should take this opportunity to meet and discuss with other people with similar interests, whether they come from Universities, R&D Labs or Industry. It is this mixture of people working environments that make discussions more fruitful (and also more lively). A limited audience should also encourage newcomers to our UNIX community to participate and get faster integration into this community.

Focusing technical presentations and discussions should allow better interactions and deeper exchange of ideas between participants. New (preliminary, crazy ?) ideas should take this opportunity for being presented, discussed and so "tested" without waiting until they are fully "ripe" (saving an implementation ?). Mike O'Dell and David Tilbrook complained "loudly" at the last Paris EUUG Conference that UNIX needed a serious "lifting" to pass next decades. Is Europe lacking new ideas ?

Covering all technical aspects related to Distributed UNIX Systems means that the workshop should not be limited to IPC's and R/NFS's (although these are important topics). One of the originality of UNIX conferences is that they are usually fun (unlike most other conferences), because of some particularly entertaining presentations. This should also be the case for the small conferences, and the theme of this workshop should allow it. We should talk, of course, about architectural and design aspects of distributed UNIX systems, kernel implementations, networking, software engineering and language aspects, algorithms for taking advantage of distribution, etc., (see the topics list in the Call

for Papers), but I would like to stress presentations and discussions on *applications* using distributed UNIX systems. And naturally applications producing "artistic" results. Isn't UNIX a piece of art in itself ?

Demonstrations

There will be no commercial exhibition. Nevertheless, if people presenting a paper wish to make a technical demonstration of their work they should be prepared for it. This form of concrete support to an oral and written presentation is heavily encouraged in the EUUG Workshops. The organisation of these demonstrations is the responsibility of the authors, of course. The only thing we need to know is your requirements. Those with such intentions should tell us pretty quickly so that we can take it into account.

Conclusion

The EUUG Manchester Workshop should be:

— less expensive,
— more technical,
— more discussions,
— more new ideas,
— more fun, and more beer...

But remember that EUUG is *You*, and EUUG conferences are *Your* conferences.
So, plan to attend and:

SUBMIT A PAPER... SEND YOUR ABSTRACT TODAY...†

A Call for Papers and preliminary announcement for the conference is in the followings pages.

† If you hesitate to send your abstract, do not hesitate to contact me.

EUUG
European UNIX† Systems User Group
Owles Hall, Buntingford, Herts. SG9 9PL, Great Britain
Tel: Royston +44 763 73039

Manchester Conference 1986

CALL FOR PAPERS
and
PRELIMINARY ANNOUNCEMENT

EUUG FALL'86 CONFERENCE
on
DISTRIBUTED UNIX SYSTEMS

Manchester (England), September 22—25, 1986

Conference structure

The next EUUG Fall Conference will be held in Manchester, England, 22—25 September 1986. It will take the form of a technical workshop dedicated to a specific topic related to Unix systems. The subject of the workshop is "Distributed Unix Systems".

Papers are solicited on the following topics.

Topics

- Design and implementation of distributed Unix systems,
- Mechanisms for distributing Unix functions and services (message passing, remote procedure call, naming, etc.),
- Inter-process communication (IPC),
- Networking: protocols, addressing, etc.,
- Remote file systems,
- Loosely-coupled Unix systems,
- Multi-processor systems,
- Parallelism,
- Applications using distributed Unix systems,
- Heterogeneity and portability,
- Distributed applications,
- Software engineering and languages for distributed applications,
- PC's in distributed Unix environments,
- Interconnecting Unix to other systems,
- Distributed computing, load sharing,
- Fault tolerance,
- Security,
- Models for distributed Unix systems,
- etc...

† UNIX is a trademark of AT&T Bell Laboratories in the USA and other countries

Tutorials

It is planned to dedicate one day to advanced tutorials on subjects related to the Conference topics. Suggested topics for Conference tutorials include:

- Technology for distributed systems: LAN (fibre optics), IPC (Sockets, Streams),
- Networking: Protocols, ISO,
- Remote File systems implementations (RFS, NFS)...

Please let us know your interests, preferences and suggestions.

Paper submission

Abstracts should be submitted to the EUUG Secretariat **and** to the Program Chairman, by ordinary and electronic mail (if possible in troff -ms form). They should include the following information:

- Title of paper,
- Author(s) name(s),
- Author(s) affiliation(s),
- Speaker name and affiliation (if more than one author),
- Mail address,
- Telephone number,
- Fax and/or telex number,
- Electronic mail address,
- Special audio-visual requirements if necessary,
- Text of abstract (in English): about 250 words.

Deadlines

- **June 20, 1986** : Abstract received by EUUG Secretariat and Program Chairman,
- **June 30, 1986** : Notification of acceptance or rejection by Program Committee,
- **August 30, 1986** : Final paper received by EUUG Secretariat and Program Chairman for publication in the Conference proceedings, (and free attendance to the Conference for the speaker).

Program Committee

Michel Gien (Chairman)
PAA/TIM
CNET
38-40, Rue du Ge'ne'ral Leclerc
92131 — Issy-les-Moulineaux
France
Tel: +33 1 45 29 62 87
Fax: +33 1 45 29 60 38
E-mail: mcvax!vmucnam!mg

Hendrik-Jon Thomassen
AT Computing
Postbus 1428
Toernooiveld
6501 — BK Nijmegen
The Netherlands
Tel: +31 80 56 68 80
Fax: +31 80 55 34 50
E-mail: mcvax!kunivv1!hjt

Neil Todd
Department of Computer Science
University of Manchester
Oxford Road
Manchester — M13 9PL
Great Britain
Tel: +44 61 273 71 21 (Ext. 5018)
Telex: 668932 MCHRVL—G
E-mail: mcvax!ukc!man.cs.ux!neil

Secretariat

Mrs. Helen Gibbons
EUUG
Owles Hall
Buntingford, Herts., SG9 9PL
Great Britain
Tel: +44 763 73039
E-mail: mcvax!euug

UNIX Security

John J. Mackin

Basser Department of Computer Science
University of Sydney

john@basser.oz
seismo!munnari!basser.oz!john

This is the first in a series of columns on the subject of UNIX security. These columns will not simply rehash back issues of the Security Mailing List, or provide general advice like 'Watch the modes of files in /dev;' rather, my aim will be to provide the system administrator with information about currently important bugs and loopholes. I plan to discuss security in a concrete, rather than abstract, fashion.

I will publish sample code to demonstrate those bugs which require programming to exploit. Such code will be verified before being published, and the affected versions of UNIX will be stated. Only by clearly identifying security holes can we foster their elimination.

Your contributions are solicited for this column. If you find a hole in your system, help to make others aware of it by submitting it for publication. Send a description of the problem, with code to manifest it if possible, to the author's ACSnet address: john@basser.oz, or by paper mail to

> John J. Mackin
> Basser Department of Computer Science F09
> University of Sydney
> Sydney NSW 2006

Please include details of the version and origin of the affected user—mode programs, kernel version, and type of hardware if that's relevant.

I will conclude this column with a description of a quite widespread security hole; it can usually be found on a fairly large portion of the boxes seen at any UNIX exhibition. Unfortunately, fewer system administrators seem to be aware of this one would hope.

This is applicable to any UNIX system that runs the popular editor 'vi.' Many systems have the temporary file preservation program '/usr/lib/ex<version>preserve' [1] setuid—root. This is the way the distributed makefile for vi installs it. The motivation for this is to keep the /usr/preserve directory not writeable by others. Vi will exec expreserve to copy the editor temporary file from /tmp to /usr/preserve if it receives a terminate signal, and so expreserve will be running with the uid of the user who invoked vi; were it not setuid, it would not be able to write in /usr/preserve unless that directory were writeable by others.

So far, this is fine. But the standard version of expreserve wants to send mail to the user, to notify them that their file was preserved. And it invokes the mail program using the library function popen, the standard version of which uses the PATH environment variable to locate the program. SO, if the user's PATH contains a 'mail' program which will be found before the system standard one (/bin/mail), it will be invoked with root privileges.

The simplest fix for affected sites is simply to make expreserve setgid, say to bin, instead of setuid—root, and set /usr/preserve to be group—owned by, and writeable by bin.

[Reprinted from AUUGN Australian UNIX systems User Group Newsletter, Volume 6 Number 3]

AWK TAWK

Ken Frame

Awk has accurately been defined as a 'pattern scanning and processing language', but this definition indicates little about its power and versatility. Most UNIX primers deal with awk from a keyboard command or shell command approach. In the commerical field of operations, awk files play a much more important part in operations than do awk processes in shell programs. Therefore most examples in this series will be in the form of an awk program file — the awk tool. This approach does not detract from the important place that shell processes have, but seeks to identify the appropriate tool for the job.

What does awk do?

Awk can do anything! Well, almost anything. Awk can reproduce text, rearrange it, print it backwards, word for word or even character for character if you want it that way. Arithmetic calculations are handled with ease, and awk contains a number of built in functions for those whose needs are more than the basic operations.

There are 3 main segments of an awk program, the beginning, the body and the end. The beginning and end segments consist of one statement each which commence with the words BEGIN and END. The body consists of one or more statements. It is not mandatory to have BEGIN and END statements and in the example below, these are not used. Statements may vary in length so that it is possible for a BEGIN or END statement to be quite extensive although neither can extend beyond one statement.

Awk performs its duties by 'pattern scanning and processing'. Each statement has two parts. The first part determines which sections of the scanned data match the pattern and ae to be processed. The second part describes the process that is to be executed. The first part, known as the pattern is optional. By default, the whole input line becomes the pattern if there is no specified pattern. Thus an awk statement consisting of:

```
{ print }
```

would print each complete line of data input.

In this article I present a very simple awk program which describes the mathematical possibilities. Consider the position of a retailer who buys in case lots and sells by the unit. The supplier's invoice for the liquor department might follow the following lines:

item	code	bottles per case	cost per case

The retailer would input the data, as in the following example.

Johnny Walker Black Label	jwb	12	250

He designs a program called grogawk to manipulate the above line (grogdata).

```
{
        ident = $1; btc = $2; each = $3                    #line 1
        print ident, btc, each                             #line 2 (debug)
        coins = int(each * 1.3/btc + .5)*2)                #line 3
        print coins                                        #line 4 (debug)
        btsell = sprintf(%6.2f, (coins / 2))               #line 5
        print btsell                                       #line 6 (debug)
        csell = sprintf("%41d",(btsell * btc * 0.95 + 1)/1)  #line 7
        cost = sprintf("%7.2f",each)                       #line 8
        print ident, cost, csell, btsell                   #line 9
}
```

Note that there is no pattern in the program. This is seen by the first character being '{' which signifies the start of the processing section of the statement. The statement consists of one or more sub—statements. A sub—statement terminates with a semi—colon or new—line. Some texts state that these characters along with the right brace '}' terminate statements, but it safer to recognize the pair of braces in a more important position. In fact, the statement is only terminated when the matching right brace is added. An attempt to terminate a statement by any other character results in an error message. Therefore I tend to think of braces enclosing statements with semi—colons and new—lines separating sub—statements.

Now, to analyze the program grogawk and the command

```
awk - f grogawk grogdata
```

Grogawk reads in data, a line at a time. Grogdata supplies the first and, in this case, only line which grogawk processes as follows:

```
ident = $1; btc = $2; each = $3                    #line 1
```

The left brace starts the processing; assign the contents of field 1 to ident, field 2 to btc (bottles per carton) and field 3 to each (cost per case). This line is not essential as the fields can be identified by their position. However it is often easier to design a program using identifiable variables than field numbers. It is certainly easier to amend a program that does not contain a mass of $1,$2,$3, etc and similarly if this program was to be adapted to a similar operation, the use of variables makes the adaptation easier.

```
print ident, btc, each                             #line 2
```

this line was inserted to test the program during the trial period. The output should mirror the input. Similarly, the line could have read print $1,$2,$3 for the same result. If all that was required was the reprinting of the entire line, a more appropriate command would be 'print $0' because $0- identifies all the fields in the line. In fact this may even be shortened to 'print' which again means print the line. As the purpose of this line is to determine the contents of the variables ident, btc and each, it is best to use the statement as in line 2. Note that each line contains a comment which commences with the character #. Comments are not mandatory, but do assist in describing the logic or purpose of the line. In this case, line 2 may be deleted when the program is running correctly.

Our merchant's pricing policy is that all items have a sale price that is a multiple of 50c. His awk program is designed to reflect this. Line 3 performs a number of operations, namely, adds a 30% markup (*1.3), divides the case price by the number of bottles per case (/btc) and calculates the number of 50c pieces in the selling price. Not wanting to reduce his 30% margin, he adds one 50c (+.5) to his calculation which is still in dollars and cents and multiplies this by 2 (*2) to count the number of coins. This in itself does not delete the decimal portion of the result but by calling for the value to be expressed as an integer (int) he now has rounded up number of coins in the sale price. The int function always truncates the value, hence the need to add 0.5 before performing the function.

```
coins = int((each * 1.3/btc + .5)*2)               #line 3
```

Line 4 prints the result of the operation, namely 55 which is produced by

```
250 * 1.3 = 325
325 / 12 = 27.0833
27.0833 + .5 = 27.5833
27.5833 * 2 = 55.1667
int (55.1667) = 55
```

```
btsell = sprintf("%6.2f",(coins / 2))                    #line 5
```

Line 5 calculates the price per bottle, namely $27.50 which is the nearest rounded up multiple of 50c from 27.0833. Unwanted digits are removed from the result by prescribing the print format. In this case the format %6.2f requires a value occupying 6 positions with 2 to the right of the decimal point. This right justifies the value which makes it suitable for printed output.

```
csell = sprintf("%41d",(btsell * btc * 0.95 + 1)/1)      #line 7
```

Our merchant will also sell by the case and allows a 5% discount but again wishing to maintain his margins, arranges his price so that instead of a 50c increment he now deals in whole dollars.

Line 7 can be read as :

*case sell = the price in 4 digit integer, (bottle price * number of bottles to the case - 5% rounded up to the next dollar).*

As in other programming languages, positioning of parentheses determine the order of the operators. In this case, the +1 occurs before the /1.

The program could have used the int function to obtain the same result, but to do so, provides not control over the format of the output. By specifying $41d all case prices are right justified within a format of 4 character positions.

```
cost = sprintf ("7.2f",each)                             #line 8
```

Finally our retailer likes to have the actual cost handy so that he can wheel and deal, so specifies that the 'each' data input at line one is printed out in a 7 character position format

```
print ident,cost,csell, btsell                           #line 9
```

The final result displays the result:

```
jwb          250.00      314          27.50
```

The ident, the cost, case selling price and bottle selling price are in the specified formats thus providing columns without having to order them by tabs.

It is likely that if our retailer was to look at that output at a later time, he would have difficulty in identifying what it meant or when it was produced. Headings, conclusions, dates and other identifiers are essential parts of programming. The role of BEGIN and END statements was mentioned briefly in this article. Can you suggest ways that the output could be made more meaningful? This is something we will consider in the next article.

[Reprinted from AUUGN Australian UNIX systems User Group Newletter, Volume 6 Number 4]

The UNIX Hierarchy

Compiled by Olle Johansson

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

guru	uses m4 and lex with comfort writes assembly code with 'cat>' uses adb on the kernel while the system is loaded customizes utilities by patching the source reads device driver source with his breakfast can answer any unix question after a little thought uses make for anything that requires two or more distinct commands to achieve has learned how to breach security but no longer needs to try
wizard	writes device drivers with 'cat >' fixes bugs by patching the binaries can answer questions before you ask them writes his own troff macro packages is on a first—name basis with Ken, Dennis and Bill

[Reprinted from Svenska Unix Systems Anv:andares F:orening (EUUG-S) Medlemsblad 1/1986]

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

RFS Architectural Overview

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

AT&T
190 River Road
Summit, NJ 07901

ABSTRACT

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

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

1. Introduction

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

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

2. RFS Goals

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

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

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

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

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

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

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

3. RFS Architecture

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

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

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

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

3.1 Client/Server

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

Any machine may be a client, server or both.

3.2 Connection Establishment

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

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

Figure 1. Remote Mount Model

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

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

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

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

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

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

Figure 2. Domain Naming Scheme

3.3 RFS Message Protocol

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

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

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

3.4 RFS File System Type

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

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

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

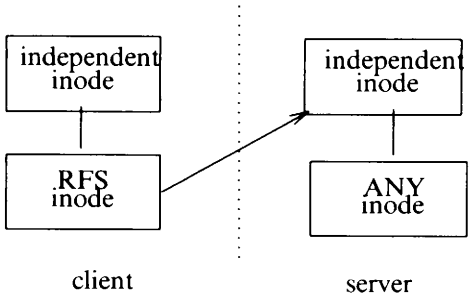

Figure 3. RFS File System Type

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

3.5 Network Independence

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

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

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

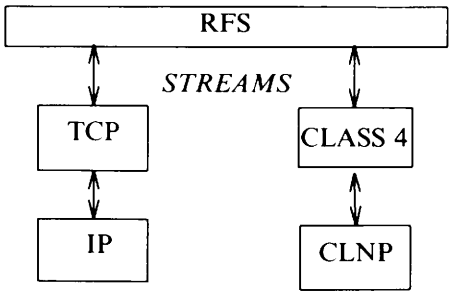

Figure 4. STREAMS Based Communication

3.6 Security

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

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

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

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

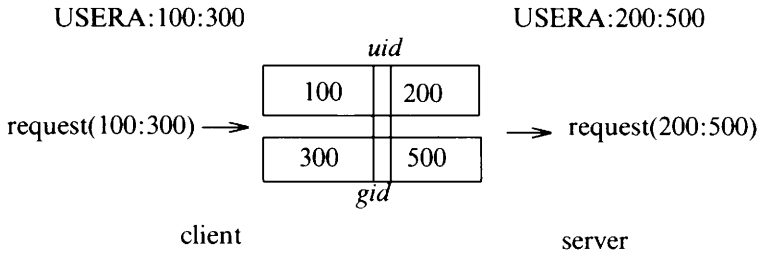

Figure 5. uid/gid mapping

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

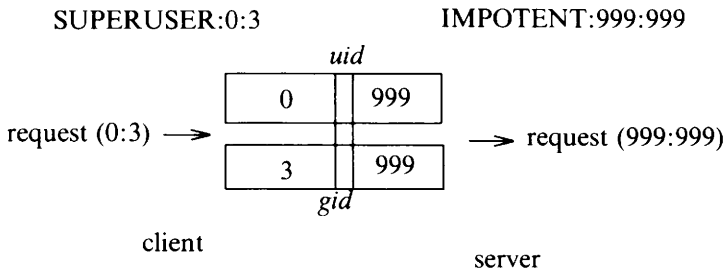

Figure 6. Super User Restriction

4. RFS Implementation

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

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

Name Server

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

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

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

Remote Mount

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

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

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

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

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

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

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

Client

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

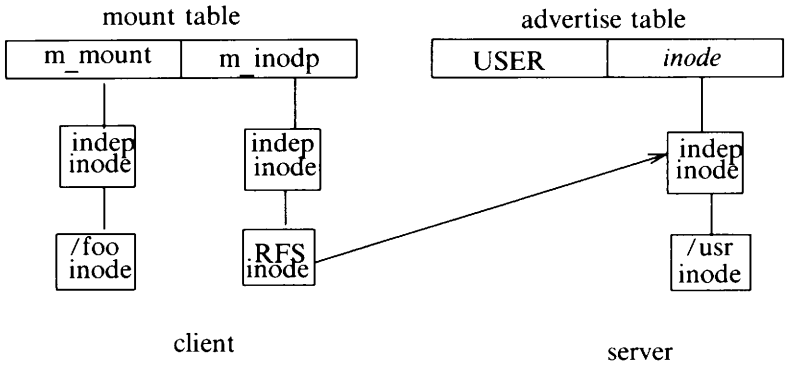

Figure 7. Remote Mount Data Structures

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

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

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

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

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

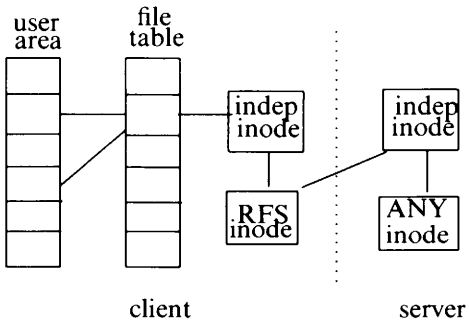

Figure 8. Open Remote File

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

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

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

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

Server

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

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

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

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

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

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

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

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

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

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

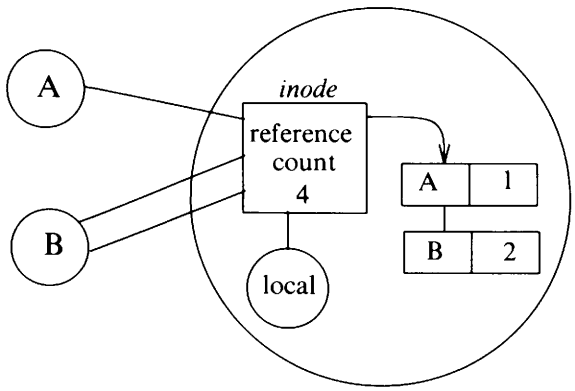

Figure 9. Server Recovery

5. Interesting Issues

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

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

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

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

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

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

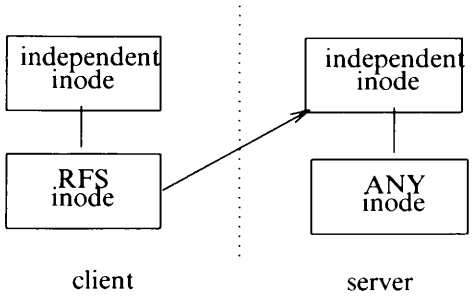

Figure 10. Remote Signal

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

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

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

Time Skew

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

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

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

6. Conclusion

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

7. Acknowledgements

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

REFERENCES

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

News from Finland
UNIX† and the polar bears

Johan Helsingius
(julf@penet)

Disclaimer:

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

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

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

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

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

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

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

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

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

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

Chairman	Johan Helsingius	Oy Penetron Ab	julf@penet
Vice-chairman	Timo Hirvonen	Oy Olivetti Ab	thi@olisf0
Secretary	Ari Kyhala	Oy Mercantile Ab	ajk@me-ncr
Treasurer	Lauri Lounasheimo	Nokia Data	
Members:			
	Par Andler	Hewlett-Packard Oy	pivi@hpuhela
	Mika Arkiomaa	Systecon Oy	
	Hannu-Matti Jarvinen	Tampere Univ. of Techn.	hmj@tut
	Jari Mantsinen	ETLA	jma@etlahp

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

Ok, ok, I'll try.

---

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

— We don't belong to the EEC. No ESPRIT, haha..

— We Finns are naturally suspicious about new ideas from 'outside'. You see, if 'they' are right, then we are wrong, and as we can't be wrong, 'they' must be!

— The Finnish industry is strongly biased towards 'heavy metal', e.g. shipbuilding, paper mills etc. This makes real-time process control a big hit. UNIX isn't.

— Another factor is that a significant part of our export trade goes eastwards. As Sheriff Reagan and his crew have decided that UNIX is a highly secret weapon, we must use proprietary operating systems or Yugoslavian V7 clones instead. Yacc!

— Finally, Sweden has gone into UNIX in a big way. Now, we can't follow every whim the Swedes do come up with, can we? Of course not, so no UNIX here, please! (By the way, do you know how to sink a Swedish submarine? Just knock on the door! Ha, ha, ha!) [Editorial Note: Did you hear that the Finnish Navy lost its submarine fleet last spring, when they installed the screen windows for summer?]

— It's cold and dark here! Anybody in need of UNIX gurus with experience handling reindeers?

Get the picture?

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

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

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

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

Cheers,

Julf

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

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

"What did you say? No polar bears?"

DKUUG since Paris 1985

Keld Jørn Simonsen
keld@diku.uucp

DKUUG
Centre for Applied Datalogy
University of Copenhagen

Changes for DKUUG since April 1985

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

DKUUG organization

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

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

Meetings

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

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

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

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

Publications

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

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

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

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

The UNIX network in Denmark April 86

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

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

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

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

Abstracts from the Florence Technical Programme

Malcolm Agnew

DB++ Database Management System

Mail: robert@hslrswi
Phone: +69 597 029798

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

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

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

Charles Bigelow

Florentine Inventors Of Modern Alphabets

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

Bigelow and Holmes
15 Vandewater Street
San Francisco
CA 94133
USA

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

Massimo Bologna

The Portable Common Tool Environment Project

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

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

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

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

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

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

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

C. Brisbois

SIGMINI Information Management System

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

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

Antonio Buongiorno

Office Data Base Services in a UNIX Architecture

Phone: +39 125 52 15 92

Olivetti DSRI/DPS
IVREA
Italy

Antonio Buongiorno
Franco Calvo
Bruno Pepino

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

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

Brian Collins

The Design Of A Syntax-directed Text Editor

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

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

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

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

This paper concentrates on the latter.

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

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

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

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

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

Pete Delaney

A Guided Tour of OSI based NFS

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

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

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

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

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

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

Winfried Dulz

System Management for a Distributed UNIX Environment

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

IMMD 7
Martensstrasse 3
8520 Erlangen
WEST GERMANY

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

William Fraser-Campbell

Implementing the NFS on System V.2

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

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

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

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

This paper will present details of the implementation.

M Guarducci

Fiore Project: Wide Band Metropolitan Area Network

Electronics Department
Florence University
Florence
ITALY

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

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

Mike Hawley

Developments at Lucasfilm

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

PO Box CS 8180
San Rafael
CA 94912
USA

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

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

Robert Heath

Adding Commercial Data Communications to UNIX

Phone: +1 513 445 6583

NCR Corp.
Columbia, South Carolina

Tlx: 851295467CZ8123Z 295467

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

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

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

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

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

Bill Joy

UNIX Workstations: The Next Four Years

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

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

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

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

Tom Killian

Computer Music under UNIX Eighth Edition

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

AT&T Bell Laboratories
Murray Hill
New Jersey
USA

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

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

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

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

S Mecenate

The IBM 6150 Executive AIX

Phone: +44 1 995 1441

IBM UK Ltd
Chiswick
London W4

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

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

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

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

Marco Mercinelli

SNAP: Restarting a 4.2 BSD process from a snapshot

Mail: ukc!cselt!marco@i2unix.uucp

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

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

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

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

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

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

Roberto Novarese

An Office Automation Solution with UNIX/MS-DOS

Phone: +39 125 52 15 92

Olivetti DSRI/DPS
Ivrea
ITALY

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

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

Philip Peake

Implementing UNIX standards

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

Axis Digital
135 Rue D'Aguesseau
92100 Boulogne
FRANCE

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

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

Dave Presotto

Matchmaker: The Eighth Edition Connection Server

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

AT&T Bell Laboratories
Murray Hill
New Jersey 07974
USA

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

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

John Richards

Software for a Graphics Terminal in C

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

University of Bristol Computer Centre
University Walk
Bristol
BS8 1TW

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

Andy Rifkin

RFS in System V.3

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

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

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

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

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

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

Russel Sandberg

Design and Implementation of NFS

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

Sun Microsystems Inc
2550 Garcia Avenue
Mountain View
CA 94043
USA

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

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

David Tilbrook

Managing a Large Software Distribution

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

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

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

Peter Weinberger

The Eighth Edition Remote Filesystem

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

AT&T Bell Laboratories
Murray Hill
NJ
USA

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

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

Lauren Weinstein

Project Stargate

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

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

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

The Florence Competition

Peter Collinson

Secretary — EUUG

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

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

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

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

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

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

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

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

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

EUUG	Unknown user group

A close second was

E	Too verbose

but the undoubted winner was

ENOTOBACCO	Read on an empty pipe

EUUG Tape Distributions

Frank Kuiper

Centrum voor Wiskunde en Informatica
Amsterdam

This is a list of all the current (June 1986) EUUG distributions. It is a general description of the available tapes. Any changes of the contents of the tapes, as well as announcements of new tapes will be placed in eunet.general and the EUUG Newsletter.

Prices of the tapes are in Dutch guilders (HFl), and do not include VAT-taxes or postage. Note also that you have to be an EUUG member (or a member of a local UUG) to obtain tapes at list prices. Non members will have to pay an extra HFl 300,- per tape.

EUUGD1 R6: UNIX V7 system, specially made for small DEC PDPs (11/23, 11/34, etc). The Kernel supports the UK terminal driver. V7 source license minimum.
Price: HFl 120,-

EUUGD2: Early Pascal compiler of the Free University of Amsterdam. V7 source license minimum.
Price: HFl 120,-

EUUGD3 R3: UNIX Networking software, 'news', and some auxiliary programs. A fairly debugged version of UUCP and X.25 support will be added if a copy of the source license agreement for at least UNIX version 7 is included.
Price: HFl 60,-
If requested, two tapes containing the major news-groups received on the Continent for the last several months is available.
Price: HFl 240,-

EUUGD4: Software tools, sampled by the Software Tools Users Group. Most of the software is written in Ratfor, for which a Fortran support tool is included. This tape is available in different formats: DEC RSX, DEC VMS, UNIVAC, IBM MVS, UNIX tar, MIT line feed format, and MIT card format (80 columns).
Price: HFl 150,-

EUUGD5: A collection of benchmark programs made up by EUUG.
Price: HFl 60,-

EUUGD6: USENIX tape, containing contributions from various UNIX System Group Members. This is a license dependent distribution: V7, V32, SIII, V6 or no license disclosure available.
Price: HFl 240,-

EUUGD7: UNIX|STAT 5.2. A collection of about 25 data manipulation and analysis programs written in C by Gery Perlman.
Price: HFl 60,-

EUUGD8: A collection of usefull software, based on the so called Copenhagen tape (EUUG Unix conference Autumn 1985). Including: the Langston binaries, hack, the rand editor, rn 4.3, shar, wm, xlisp, Emacs, Netnews and much more.
(Also: Yacc-pcc, by: J A Dain, Dept of Computer Science, University of Warwick. For Yacc-pcc you need at least AT&T V7 source licence. This part is only included on request)
Price: HFl 120,-

EUUGD9: A collection of usefull software, based on the so called Florance tape (EUUG Unix conference Spring 1986). It consists of the following:

INDEX	Indexes of articles on the net
RFC	"Request For Comments" - proposed standards papers
mh-6.4	Berkeley's enhanced UCI Mail Handling system
emacs17.49	Richard Stallmans GNU emacs editor
ned	The rand editor, version E17
scame	An Emacs-like editor
teco	The editor known from DEC systems, for VAX and 6502
netnews	The source needed to run news, 2.10.3-alpha
rn	Larry Wall's program to read the news stuff, version 4.3
statistic	The UNIX|STAT statistical package from Gary Pearlman
forth	A compiler for the forth language for VAX BSD systems
xlisp	A lisp interpreter, version 1.4 by David Betz
trc	Daniel Kary's expert system building package
terminfo	A library of rutines handling screens thru TERMCAP
vsh	Visual shell 4.2, for various UNIX systems and machines
window	A windowing system
less	A paginator a la more or pg
rfs	The Remote File System from T. Brunhoff of U. of Denver
rpc	The Remote Procedure Call system from SUN Microsystems
kermit	The C-kermit transmisson program version 4C(057)
langston	The Langston games for 4.2 BSD VAX and SUN systems
hack	The famous rogue-like game, 1.0.3
rogomatic	The automatic rogue game
battlestar	An adventure-like game
galaxy	Yet another game

Price: HFl 150,-

EUUGD10: MMDFIIb. Multichanel Memo Distribution Facility (version IIb). This is a powerful, domain oriented mail system with access control and the ability to communicate over a variety of network systems including TCP/IP, JANET, UUCP, PHONENET, etc. It has been ported to a variety of UNIX's including but not limited to 4.[123]BSD, 2.9BSD, System III/V on a variety of different hardware. You should first obtain a license agreement by sending a message to euug—tapes@mcvax. Return the signed license with your order.
Price: HFl 90,-

EUUG Tape Distributions Order Form

If you want to order any tape, please write to:

EUUG Distributions
c/o Frank Kuiper
Centrum voor Wiskunde en Informatica
Kruislaan 413
1098 SJ Amsterdam
The Netherlands

For information only:

Tel: +31 20 5924056 (or: +31 20 5929333)
Telex: 12571 mactr nl
Internet: euug-tapes@mcvax (or: frankk@mcvax)

Please note that for distributions D1, D2, D3 and D4 (and in some cases also for D8) a copy of your source license agreement with AT&T for at least UNIX version 7 should be enclosed.
Note also that you have to be an EUUG member (or a member of a national UUG) to obtain tapes at list prices. Non-members will have to pay HFl 300,- per tape extra as handeling fee. Please enclose a copy of your membership or contribution payment form when ordering. All tapes come in tar format, 1600 bpi. 800 bpi is possible on request. Tapes and bill will be sent separately.

Name: .....

Company: .....

Address: .....

.....

.....

I would like to order the following:

.....

.....

.....

.....

EUUG (or national UUG) membership form enclosed? Yes / No
Copy of AT&T source license enclosed? Yes / No

"I declare to indemnify the European UNIX systems User Group for any liability concerning the rights to this software, and I accept that EUUG takes no responsibilities concerning the contents and proper function of the software."

Signature ..... Date.....

The Secretary
European UNIX† systems User Group
Owles Hall
Buntingford
Herts. SG9 9PL.
Tel: Royston (0763) 73039