

AUUGN

Australian Unix systems

User Group Newsletter

Volume 8

Number 3-4



The Australian UNIX* systems User Group Newsletter

Volume 8 Number 3-4

August 1987

CONTENTS

AUUG General Information	3
Editorial	4
Softway Advertisement	5
Adelaide UNIX Users Group Information	6
Measuring Database Performance using the TP1 Benchmark	7
Finland: Birch and Boat	21
From the <i>login:</i> Newsletter - Volume 12 Number 4	28
Call for Papers: POSIX Portability Workshop	29
Call for Papers: Winter 1988 USENIX Conference	30
Call for Papers: Summer 1988 USENIX Conference	31
Computer Graphics Workshop	32
Multiple Programs in One UNIX Process	33
tar vs. cpio	39
How to Write a UNIX Daemon	43
Book Review: The Design of the UNIX Operating System	50
Book Review: A C Reference Manual	52
UUNET Progress Report	53
EUUGN Spring 1988 Conference Announcement	55
From the EUUGN Newsletter - Volume 7 Number 1	57
Unix Conference Reports	58
Atlanta Usenix, June 1986	58
The Manchester Competition	74
Uniforum, January 1987	80
Notes on the Birth of the UNIX Cult	83
The X/OPEN show in Luxembourg	91
The CV Macros	92
GKS in C++	93
An NRS Processor in C and the Future	105

From the EUUGN Newsletter - Volume 7 Number 2	108
Packets vs. Circuits, in Two Centuries	109
Music: a Troff Preprocessor for printing music scores	112
An Overview of the Native Language System	129
Grouse: Messages and Prompts in Programs	138
Another Proposal for a News Scheme	148
EUUG	150
Progress of ANSI/ISO C Standardisation	152
X/OPEN - What, Who, Why, When	157
EUnet	159
UNIX Clinic	162
Review of POSIX	164
Letters to the Editor	167
AUUG Membership Categories	171
AUUG Forms	173

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

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

AUUG General Information

Memberships and Subscriptions

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

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

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

General Correspondence

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

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

ACSnet: auug@munnari.oz

AUUG Executive

Ken McDonell, President

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

Robert Elz, Secretary

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

Chris Maltby, Treasurer

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

Chris Campbell, Committee Member

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

Piers Lauder, Committee Member (Newly Elected)

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

John Lions, Committee Member

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

Tim Roper, Committee Member

timr@labtam.oz
Labtam Limited, Victoria

Next AUUG Meeting

The next meeting will be held in Melbourne during February 1988.
Further details will be provided in the next issue.

AUUG Newsletter

Editorial

I am disappointed that the majority of this issue is reprints from the USENIX and EUUG Newsletters. I would prefer that it had more Australian content. I will continue to encourage people to write articles for the Newsletter and hope this produces a better result in future issues. Please remember that articles do not have to be about UNIX itself but applications that run under UNIX.

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

AUUGN Correspondence

All correspondence regarding the AUUGN should be addressed to:-

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

ACSnet: auugn@monu1.oz

Phone: +61 3 565 4754

Contributions

The Newsletter is published approximately every two months. The deadline for contributions for the next issue is Friday the 16th of October 1987.

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

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

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

Advertising

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

Mailing Lists

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

Disclaimer

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



for

- UNIX System V
- Documentor's Workbench 2.0
 - and various back-end drivers
 - PostScript support of plain text
 - support for graphs and images
- Ports & Device Drivers
- Intelligent Benchmarking
- SUN-III (ACSnet) + installation
- Biway - Bi-directional modem software for System V and 4bsd
- Courses:
 - Beginner's Workshop
 - Fast start to UNIX
 - System Administrators' workshop
- Technical Backup
 - and all sorts of interesting software development.

Softway Pty Ltd. (Incorporated in NSW)
120 Chalmers St, Strawberry Hills, NSW.
PO Box 305, Strawberry Hills, NSW 2012.
¶ (02) 698 2322 Fax (02) 957 6914

Adelaide UNIX Users Group

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

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

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

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

Dennis Jarvis,
Secretary, AdUUG.

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

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

MEASURING DATABASE PERFORMANCE USING THE TP1 BENCHMARK

Ken J. McDonell

Department of Computer Science
Monash University
Clayton, Victoria 3168, AUSTRALIA

Acsnet: kenj@moncsbruce.oz

ABSTRACT

This note reports on some performance experiments conducted with a commercially available relational database management system (let's call it DBMS-R) in conjunction with the TP1 benchmark[1]. These tests are of particular interest given the popularity of TP1 as a *de facto* standard for measuring on-line transaction processing throughput. This paper assumes the reader is familiar with the TP1 benchmark; full details may be found in[1].

In all cases, the tests were run on an unloaded Unix¹ machine in multi-user mode (with the usual assortment of daemons, especially the Ethernet ones). Several Unix machines were used, all from the one vendor's model range; they shall be referred to as Model-1, Model-2 and Model-4 (model numbers crudely approximate to relative raw performance).

Except where stated to the contrary (for some Model-4 tests), the filesystems all had a default configuration with a block size of 16K bytes.

The same brand of disk drives was used in all tests.

1. System and Database Configurations

1.1 Model-4

The Model-4 processor had 128 Mbytes of real memory, and 7208 buffers in the file system cache.

After several experiments, the following database configuration was used with logging, the DBMS-R system catalogs and the relations spread across 4 physical disk drives. Within the parameter set explored, this configuration gives the best performance amongst those deemed realistic and "honest".

1. Unix is a trademark of AT&T Bell Laboratories.

Relation	Size	Access Method	Disk	Filesystem
DBMS-R catalogs			disk1	default
transaction log			disk3	default
account	100000	hash unique	disk2	2K block size
teller	1000	hash unique	disk1	2K block size
branch	100	hash unique	disk1	2K block size
history	0-21500	unstructured	disk4	default

Note that the filesystems for the randomly accessed relations have been reconfigured to have block sizes of 2K bytes; refer to Point 8 in Section 4 for a full discussion as to why.

1.2 Model-2

The Model-2 processor had 64 Mbytes of real memory, and 2293 buffers in the file system cache.

Based upon the Model-4 experiences and a small number of further experiments, the following database configuration was used with logging, the DBMS-R system catalogs and the relations spread across 3 physical disk drives.

Relation	Size	Access Method	Location
DBMS-R catalogs			disk0
transaction log			disk1
account	100000	hash unique	disk2
teller	1000	hash unique	disk2
branch	100	hash unique	disk0
history	0-15400	unstructured	disk1

1.3 Model-1

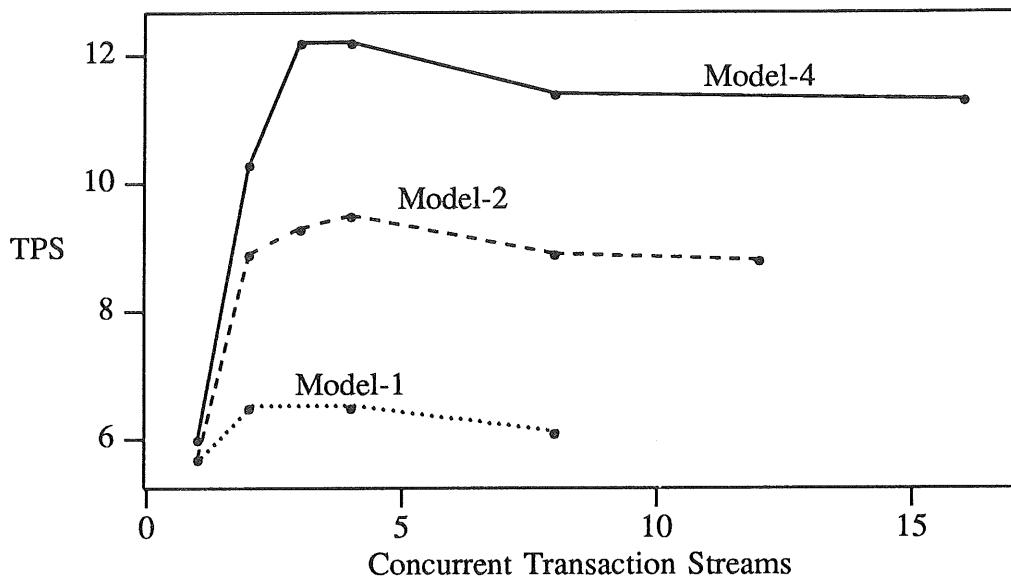
The Model-1 processor had 32 Mbytes of real memory, and 1146 buffers in the file system cache.

The following database configuration was used with logging, the DBMS-R system catalogs and the relations spread across 2 physical disk drives.

Relation	Size	Access Method	Locatio
DBMS-R catalogs			disk0
transaction log			disk1
account	100000	hash unique	disk0
teller	1000	hash unique	disk0
branch	100	hash unique	disk0
history	0-8200	unstructured	disk0

2. TP1 Performance

The measured mean TP1 performance (in transactions per seconds, TPS) with varying degrees of concurrency is shown in the following graph.



Peak TP1 throughput² is as follows.

CPU	Concurrency	Peak TP1 (TPS)	Relative to Model-1
Model-4	3	12.2	1.87
Model-2	4	9.5	1.46
Model-1	2	6.5	1.00

3. Performance Analysis and Other Tests

Across the hardware range different resources are evidently limiting TP1 throughput. The following numbers represent **average** figures gathered using *mpstat* and *dkstat* on a repeated run for the optimal degree of concurrency. Note that the column headed "Disk Xfers" is for the most heavily used drive only.

CPU	Concurrent Streams	Processor(s) % Idle Time	System Calls (per sec)	Context Switches (per sec)	Disk Xfers (per sec)
Model-4	3	43	1391	308	24, disk2
Model-2	4	48	1079	267	18, disk2
Model-1	4	13	774	134	16, disk0

Scrutiny of the *mpstat* statistics for the Model-1 reveal that it is running close to maximal CPU utilization, with bursts of high CPU idle time and near-peak disk rates coinciding with *sync* activity. Short of either spreading the disk buffer cache flushing more uniformly with time, or re-engineering some major DBMS-R component (e.g. the IPC mechanism or the lock manager), little improvement can be expected. However, better throughput should be

2. There may be some marginal improvement at degrees of concurrency between the measured points, but near the optimal throughput, e.g. 3 concurrent transaction streams for a Model-1 or 5 concurrent streams for a Model-2.

achievable because the very high system time (48%, compared to 40% user time) suggests inefficient system call patterns or poorly implemented system code.

At the other extreme, the Model-4 has plenty of unused CPU capacity and disk bandwidth, but the throughput is being constrained by a “convey” phenomenon caused by lock conflicts for the last page of either or both the transaction log and the “history” relation.

Further insight into TP1 performance may be gained from consideration of the following hypothetical transactions,

T1: Amend a tuple in relation R where the key = K

T2: Append a tuple to relation R

A TP1 transaction is composed from a set of smaller updates each being similar to either T1 or T2. Since T1 and T2 are both simpler than TP1, studies of T1 and T2 are more easily conducted, but the results may be used to predict maximal TP1 performance.

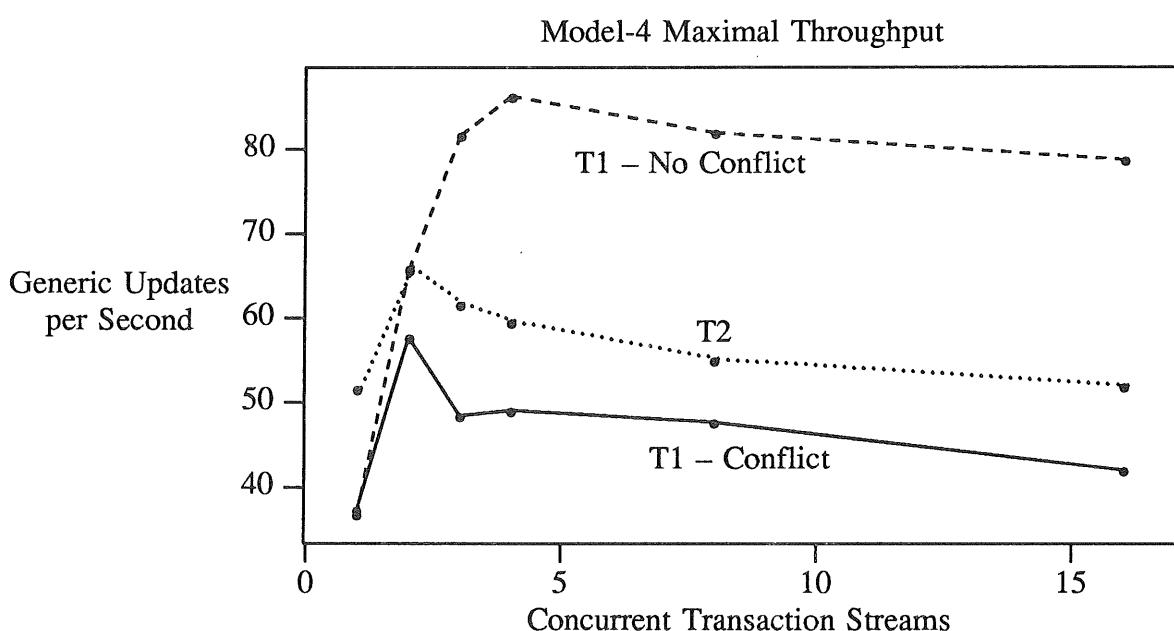
Initially we are interested in minimizing disk activity to concentrate on other performance factors – consequently all T1 and T2 tests run with transaction logging *disabled*. For T1 we can further minimize disk I/O by using a hashed access method. For T2 the unstructured access method is chosen to model transaction logging activity, and possibly the updating of the TP1 “history” relation.

If one stream of identical T1 transactions is run, statistically reliable measures may be made without any contention. Running several concurrent T1 streams allows investigation of throughput limits in the following interesting cases,

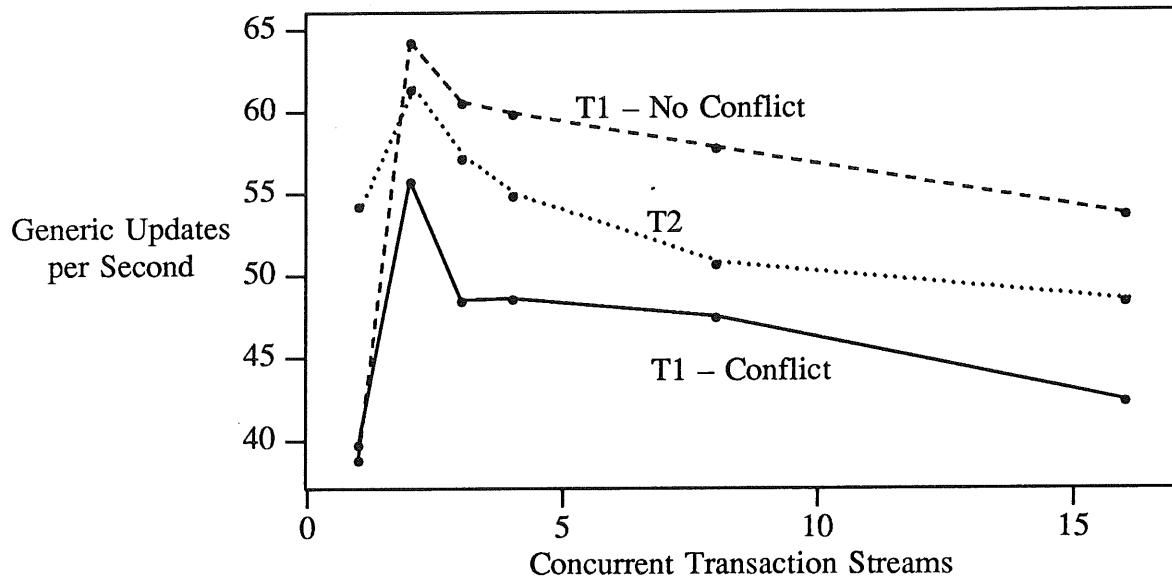
1. No conflict between transactions (all streams use a different value of the key K chosen to ensure that the corresponding tuples map to different physical database pages).
2. Pathological conflict (all streams use the same key K).

For T2 transactions, multiple streams updating the same relation are always in conflict over the lock for the last physical page of the relation.

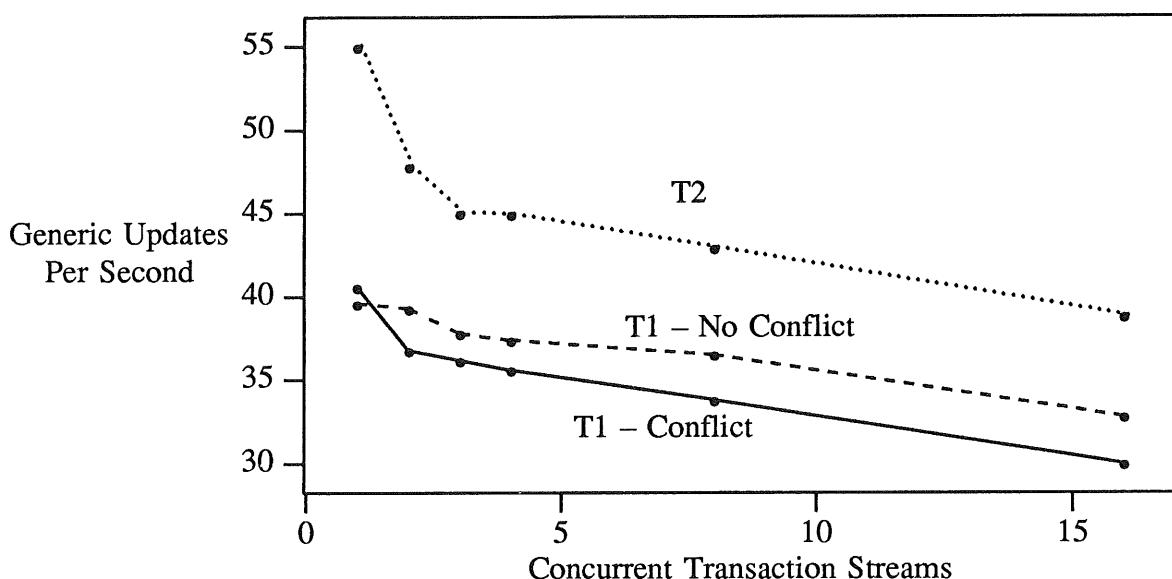
The following graphs illustrate the maximum achievable throughputs for T1 (with and without conflict) and T2.



Model-2 Maximal Throughput



Model-1 Maximal Throughput



Since each TP1 transaction consists of 2 T2 sub-transactions and 3 T1 sub-transactions (with conflict varying from none to slight), the following upper bounds on TP1 performance can be computed. These bounds are important because they are based upon no logging and minimal disk I/O – we are measuring principally the DBMS passage time per transaction, and in particular the peak rate at which the lock manager can handle transactions and resolve conflicting lock requests. Even if ideal situations prevail during a TP1 run and the additional logging and disk I/O activities can be overlapped with processing of concurrent transactions, the TP1 throughput cannot exceed these upper bounds.

CPU	TP1 Minimum Time	TPS Upper Bound	TPS observed	
			Absolute	% Bound
Model-4	$2/60 + 3/75 = 0.073$	13.6	12.2	89
Model-2	$2/57 + 3/55 = 0.089$	11.2	9.5	85
Model-1	$2/45 + 3/37 = 0.13$	7.9	6.5	82

4. Benchmarking Methodology

Several “pitfalls” and problems associated with TP1 measurements were uncovered. This list should be used as a checklist during TP1 measurements for other database management systems, and to verify the extent to which competing performance figures may be honestly compared.

1. For TP1 all tuples should contain 100 data bytes (except for the “history” relation which has 50 data bytes per tuple). Test databases with tuples of non-standard size can produce significantly different performance.
2. The size of each relation is also defined for a 100 TPS system to be as follows.

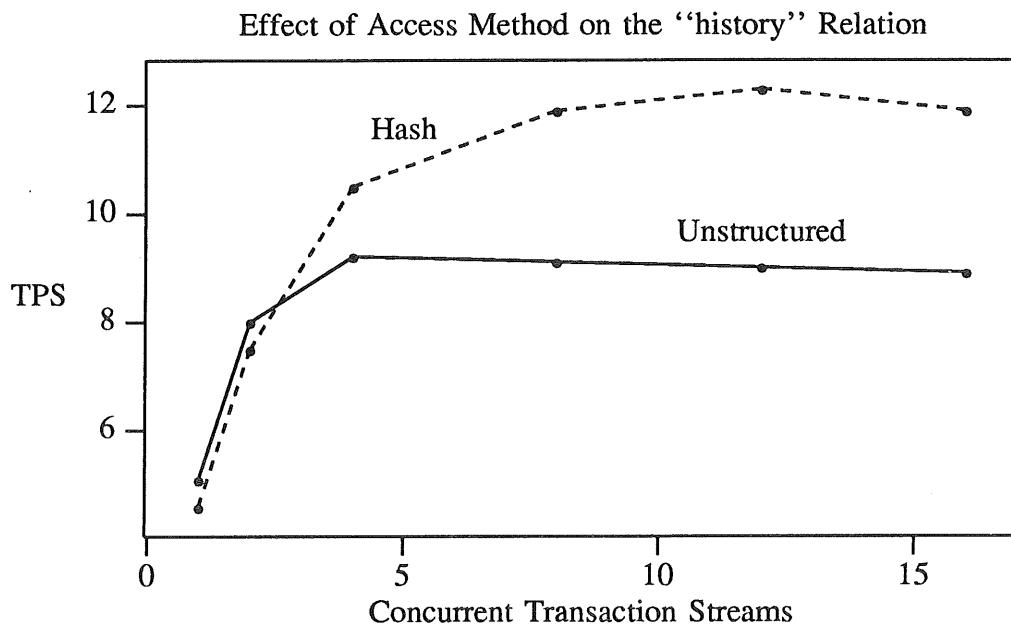
Relation	Tuples	Size
branch	1,000	100 Kbyte
teller	10,000	1 Mbyte
account	10,000,000	1 Gbyte
history	200,000,000	10 Gbyte

For a 10 TPS system (e.g. any machine in the range under investigation), these numbers should be divided by 10. However this leads to several problems.

- a. The “account” relation is 100 Mbytes of data. To achieve acceptable random access a relatively low space utilization is required (perhaps as low as 40% for some implementations), so this can easily require 200 Mbytes – a little too big for comfort, as the test database loading may take several days! Pragmatic considerations then typically reduce this relation to 100,000 tuples (1/100 scaling), justified by assertions that performance for the larger relation size would be comparable; this is reasonable if access time for “accounts” tuples is not the limiting performance factor or the access method provides access times independent of relation size (e.g. a good hashing addressing scheme) and provided there is no “cheating” (e.g. the whole database could, in theory, be loaded into the available real memory on even the smallest system in the range under consideration).
- b. There is no published evidence to suggest *anyone* accumulates 1 Gbyte of “history” data (for a 10 TPS system) before they start the TP1 benchmark. Common approaches include starting from zero, or some arbitrary token number (e.g. 10000) of tuples.
3. The organization of the “history” relation is subject to considerable variation. Two possibilities were investigated.
 - a. A sequential file. Realistic, but causes increased lock contention for the last physical page because *every* TP1 transaction must append a tuple to this table.
 - b. A hashed file. Reduces physical page lock conflict, but this is not a rational way to build a chronological record of updates. Pragmatic issues that must be addressed

include periodic reorganization (the growth is unbounded and eventually the pages become so full that the increased update times dominate total transaction time, thereby negating all gains from reduced lock contention), concurrent sorting and archival of tuples to some non-volatile storage (no attempt has been made to include this overhead in the DBMS-R measurements).

The following graph illustrates the effects of this choice. All configuration parameters are as specified in Section 1.1, except the default, rather than tuned filesystem was used and in the hashed case the “history” table was initially loaded with 10,000 dummy tuples at a fillfactor of 17% (after the last run there were 28,000 tuples and the fillfactor was 50%).

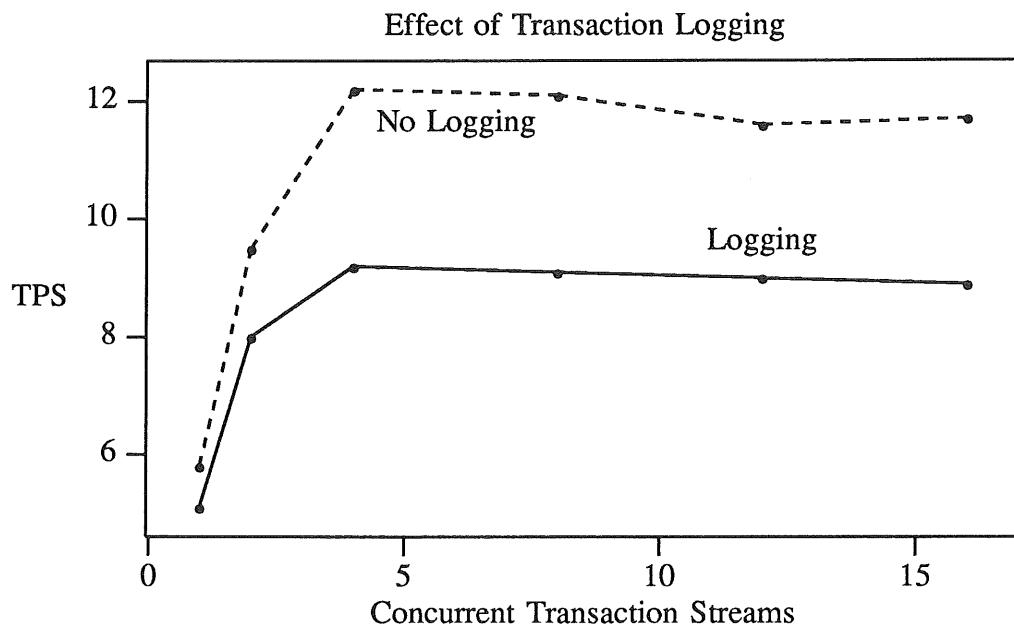


Despite the apparent better performance of the hashed scheme, all other tests reported in this document use the unstructured scheme because this organization could be sustained over an extended period without expensive reorganization.

Elsewhere, some TP1 results have been gathered using *several* “history” relations (e.g. partitioned by branch number) to spread the activity and hence reduce the contention conflict. This scheme is technically feasible and perfectly acceptable, but was not employed in these tests because no performance improvement could be expected all the while transaction logging was imposing a second convoy regime upon transaction execution.

4. Make sure transaction logging is enabled. This is **not** an optional part of TP1.

The following graph illustrates how impressive, but bogus, performance can be achieved by disabling transaction logging. Apart from logging, all configuration parameters in both runs are as specified in Section 1.1 except the default rather than tuned filesystem was used.



5. In the absence of any special precautions, concurrent TP1 transactions can deadlock when share locks are promoted to exclusive locks for each amended tuple in “account”, “teller” and “branch”. TP1 testbeds must ensure deadlock is either prevented or aborted transactions are resubmitted.

In a more general vein, the transaction implementation should include an error handling mechanism that at least detects when an update is not completed as expected.

6. Measuring TPS rates over very short times produces a “burst” TPS rating that cannot be sustained. In such a short interval, all writes may have been cached and *sync* may not have run – this makes disk writes appear much cheaper than they really are! For example if each stream consisted of 100 TP1 transactions and the degree of concurrency varied between 1 and 16 streams, then the elapsed times (across a Model-4 and Model-1) would be in the range 10 seconds to 5.4 minutes. Figures collected in this manner cannot legitimately be compared. For all results presented in this report, the aggregate number of transactions across all concurrent streams remains constant (1536) and elapsed running times are in the range 2.5 to 5.0 minutes.

Similarly, there is some freedom in the interpretation of the TPS rate, as follows

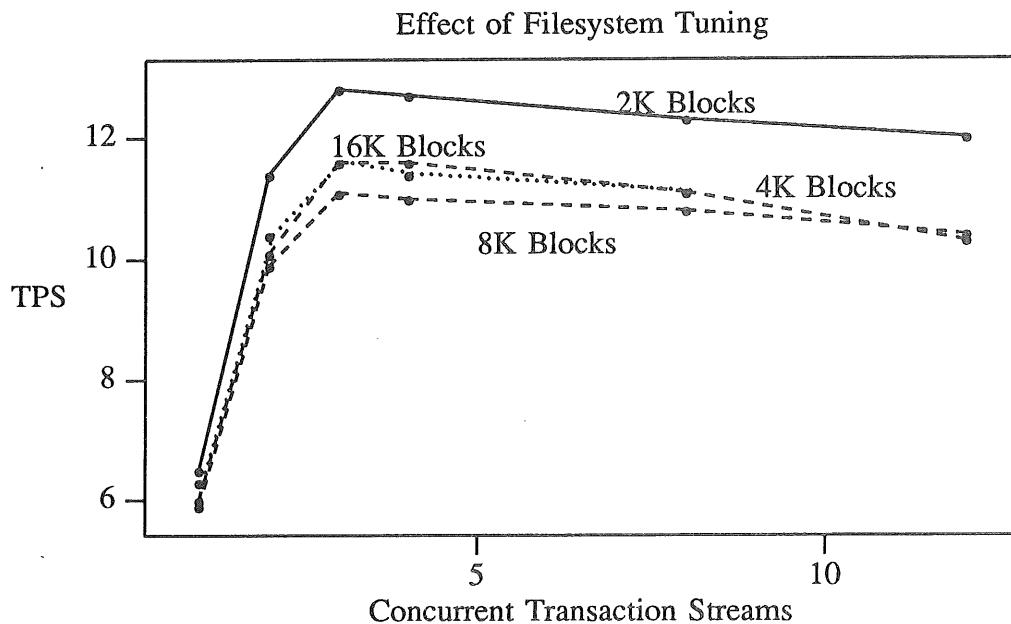
1. The number of transactions completed, divided by the sum of the running times for each transaction stream.
2. The number of transactions completed, divided by the total running time for all transaction streams (from the start of the first one to the end of the last one).

The former measure produces marginally higher values, but is also more representative of steady-state transaction processing. This is the measure used throughout this report.

7. The TP1 throughput varies dramatically with the degree of transaction concurrency. Throughout these experiments, this parameter has been considered an independent variable whose value may be chosen to maximize TP1 throughput for a particular choice of all other configuration parameters. Not all TP1 measurements performed by others are quoted in this way, so comparisons may be misleading.

Justification for choosing the degree of concurrency to maximize TP1 throughput is relatively straight forward, based upon an application architecture in which a communications front end process (or processes) manages the terminals, gathers transaction details and submits transactions to one of N queues. Each transaction queue is serviced by a dedicated server which runs one transaction to completion before starting the next transaction. Such a scheme supports fluctuating transaction arrival rates with *constrained* DBMS concurrency to achieve maximal throughput.

8. Tuning the UNIX filesystem can make a significant difference. Transaction logging and the “history” relation are both fundamentally write-only sequential files with small logical record sizes – big filesystem blocks help here. The other relations are all subject to random read and re-write, again with small logical record sizes – small filesystem blocks are optimal. The following graph illustrates the effects of this tuning for a Model-4 configuration (see Section 1.1 for parameters) as the filesystems containing the randomly accessed relations are varied from the default 16K block size configuration through to the optimal (for TP1) 2K configuration.



9. Care must be taken to see what DBMS tuning options have been invoked. For TP1, query optimization is irrelevant, but selection of other parameters can have a major impact, e.g. choice of lock granularity, lock promotion scheme, random access storage method, database buffer cacheing, etc.

Relevant options for DBMS-R throughout these tests are,

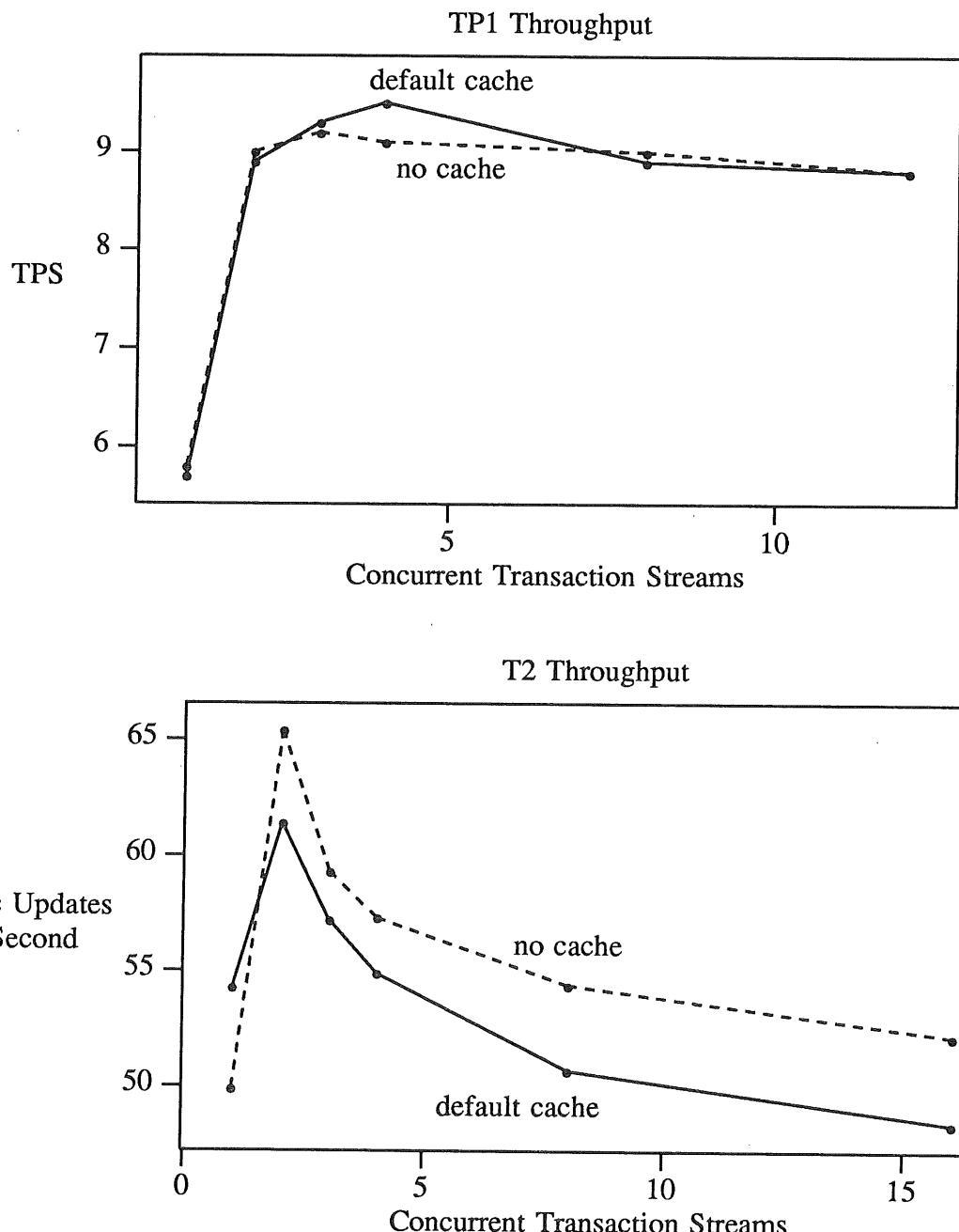
- Enable transaction logging.
- Automatically promote read locks to exclusive locks at the time of granting.
- Explicitly call the DBMS to delimit each TP1 transaction.
- Precompile queries to minimize parsing and optimization overheads.

Other tuning options may have obscure effects. For example, the number of data pages cached in each DBMS-R process may be controlled. The local cache may reduce data page reads, but incurs a synchronization overhead as additional locks for cached pages must be acquired and released. The following graphs illustrate the effects of setting the

number of cached pages to zero on a Model-2 for the TP1 and T2 runs.

Reducing the local cache size to zero improves T2 performance when there is more than one stream of transactions, since consecutive appends within a single stream will be directed to *different* data pages and page cacheing incurs the lock overhead for no gain. When there is no concurrency, the poorer performance results from repeated reads of the same (last) page of the relation that is avoided when cacheing is employed.

The situation for TP1 is less clear. No obvious explanation exists for the poorer performance in the un-cached case at near optimal levels of concurrency.



10. The original TP1 benchmark specification includes a number of aspects that are often ignored due to difficulty in implementation or inability to support the relevant facility. None of these factors have been included in the measurements presented in this report.

- Terminal I/O. Assumed to be in block mode, 100 bytes (10 fields) of input (the transaction details) and 100 bytes of output (response).
- X.25 communications between teller terminals and the host.
- 15% of transactions are for accounts held at branches *different* to the branch (teller) at which the transaction is generated.
- Duplexed transaction logging.
- At least one second response time for 95% of the transactions.

5. Making it All Go Faster

Faced with ample CPU power and disk bandwidth, but comparatively low TPS rating, it becomes necessary to identify where and how transactions are in conflict in such a manner that serious interference results. To add to the gloom, system CPU usage is always significantly higher than user CPU usage. Possible candidates include

1. Excessive system calls in either the DBMS-R process or the run-time library attached to the application process.
2. Poor IPC protocols and/or implementation between the application and the DBMS-R processes.
3. The DBMS-R lock manager.

5.1 IPC

By surgically implanting a monitor routine below the DBMS-R run-time library in an application process, it was possible to gather information on the pipe-based IPC protocols between the DBMS-R process and the T2 sub-transaction. This revealed the following facts.

- All messages are 256 bytes long.
- After the initial hand-shaking (3 messages), there are 2 messages (one send, one response) for each embedded query statement. Statement precompilation carries a further overhead of 2 messages the first time the statement is executed.

This would suggest approximately 14 messages per TP1 transaction.

Using the MUSBUS[2] *context1* test as a basis, two processes running on a Model-2 are able to exchange 20,000 messages in 34.4 seconds (1.2user + 21.5sys). Message exchange alone would therefore limit TP1 performance for *one* transaction stream to a peak rate of 40 TPS.

The overhead in message processing is clearly not the limiting resource, although in conjunction with some other computationally intensive activity (e.g. the DBMS-R process or the lock manager) it may well represent a major contributor to the high relative system CPU time and lower than expected transaction throughput.

5.2 System Call Behaviour

Using a profiled kernel on an Model-2, 4 streams of T2 transactions were run. The following system call frequencies were observed and expected (only calls with frequencies over 500 shown).

System Call	Frequency		Explanation
	Observed	Expected	
ioctl	17267	7000	Based upon 3 lock requests and 1 lock release per T2 transaction.
lseek	6422		
write	5197	4700	3100 for pipe-based IPC messages, 1600 disk writes.
read	3423	3100	Pipe-based IPC messages.
sigblock	3333		
close	576		

Worthy of note here is the unexpectedly higher number of *ioctl* calls, and large numbers of totally unexpected calls to *lseek*, *sigblock* and *close*.

5.3 Kernel Profiling

In an attempt to identify the reasons for such high system CPU time, three tests were performed with a profiled kernel.

Case A T2 with 1 and 4 concurrent streams on a Model-2. The results with one concurrent stream are not very interesting (each CPU is 50% idle), so the table below includes only the 4 stream case.

Case B TP1 with 3 concurrent streams on a Model-2.

Case C TP1 with 4 concurrent streams on a Model-4.

The following table summarizes the main contributors, with resource consumption shown in absolute terms and as a percentage of the system CPU time accumulated below the system call entry (i.e. excluding idle and interrupt handling time).

The times were extracted using *gprof* and classified as follows.

DBMS-R Lock Driver

Everything in *ioctl* and below; there are effectively no calls to *ioctl* for devices other than the DBMS-R lock manager.

File and Pipe I/O

Calls to *rwuio* via *read* and *write* are aggregated, and then usage attributed to file or pipe I/O according to the frequency of calls to *vno_rw* and *pipe_rw* respectively.

Case	DBMS-R Lock Driver	Pipe I/O	File I/O
A	23.58 (40%)	12.94 (22%)	8.50 (14%)
B	61.55 (23%)	87.52 (33%)	52.18 (20%)
C	82.75 (23%)	134.47 (38%)	64.62 (18%)

The execution time per request in the DBMS-R lock manager depends upon the degree of lock conflict and varies from 0.5 msec/call in Case A (no concurrency), through 1.1 msec/call (Case A with 4 streams and Case B), to 1.5 msec/call in Case C. This suggests that for TP1 the minimum time that a non-shared lock may be held (e.g. exclusive access to the transaction log) is of the order of 3 msec, or some 350 lock request-release cycles per second. This cannot produce the convoy effect,

because the observed peak rate in the order of 12 TPS means the lock on the resource that is causing the convoy must be held for about 90 msec by each transaction. Possible explanations include,

1. The DBMS-R process does an enormous amount of work between acquiring the lock and releasing it.
2. There is some major logical flaw in the lock manager that is causing the process making a lock request to be blocked for no apparent good reason, thereby adding considerable real-time delay to each lock request and/or release that appears as CPU idle time, rather than accumulated CPU time in the lock manager.

The first explanation seems more plausible, but it has not been possible to prove or disprove either hypothesis.

6. Concluding Recommendations

The following issues must be addressed before DBMS-R TP1 performance on the studied machines can be expected to demonstrate dramatic improvement.

1. The cause of the “convey” mechanism must be clearly identified. Until the passage time for use of this critical resource is reduced, or removed the Model-2 and Model-4 will continue to demonstrate throughput that degrades above a concurrency level of about 4 and at the same time leaves considerable CPU idle time (note that disk bandwidth will not become an issue of any relevance unless there is a significant increase in overall transaction throughput).

Some possible techniques to help in this area include,

- Multiplex the transaction log, e.g. one log per DBMS-R process with centralized co-ordination (system-wide log) only occurring when the DBMS-R process starts-up and shuts-down, rather than at each transaction commit.
 - For sequential files (e.g. a unstructured relation, or the transaction log) consider using atomic append-writes (as supported in the kernel, i.e. open the file with mode O_APPEND) in preference to DBMS-level lock protocols and *lseek* before each write.
 - Avoid deadlock detection overhead for blocked requests to lock the transaction log – transactions blocked here should be waiting for a short time with no chance of deadlock, unless DBMS-R is in the midst of a pathological crisis from which it is unlikely to recover!
2. Improved pipe performance or a more efficient IPC mechanism between the DBMS-R and the application processes. By itself, this cannot be expected to improve performance (except on the Model-1), because this IPC activity is almost certainly outside the scope of the interaction responsible for the convey phenomenon. However, if the serial execution imposed by the convey were relaxed, then system CPU time spent supporting pipe I/O would become a major component limiting throughput as CPU utilization nears saturation.

At a more general level, this study has highlighted a number of potential weakness with current TP1 specifications and implementation that should act as a warning to vendors and purchasers alike – be very careful if you are using TP1 results to influence decisions in respect of critical resources like people and money.

References

1. Anon et al., A Measure of Transaction Processing Power, *Datamation*, Apr. 1, 1985, 112-118.
2. Ken J. McDonell, Taking Performance Evaluation Out of the "Stone Age", *Proc. USENIX Summer Technical Meeting*, Phoenix, Arizona, June, 1987, 407-417.

Finland: Birch and Boat

Rob Pike

ABSTRACT

Trip report from the European Unix User's meeting in Helsinki, Finland, and related events.

Minix

First stop was Amsterdam, to visit Andy Tanenbaum and chat about his new V7-compatible system, Minix. Minix's purpose is pedagogy: Tanenbaum has written an undergraduate textbook using Minix as a complete system for study, much like Lions's V6 books from the late 70's. Minix is successful in the classroom — the benefits of study by example are well known. Students are given lab assignments like "add symbolic links" or "write a device driver for the Epson printer." Also, Tanenbaum feels (and I agree) that operating systems texts have tended too much towards the theory and not enough towards the practice. As he puts it, they'll devote pages of queuing theory to round-robin scheduling when a few lines of code and some common sense could do as well.

Minix is also successful as a book, and as a system. Prentice-Hall sells the complete source code for about \$100, and a group of avid Minix hackers has come to life almost overnight. Tanenbaum is delighted.

This might be a good time to dispel some of the misconceptions floating around about Minix. First, it is decidedly *not* V7. The kernel has been completely rewritten and redesigned, and at least some of the user-level commands have also been rewritten. A number of commands are missing, the most striking being *ed*. Also, because of some simplifications in the kernel, a number of system calls are missing as well, although none of them too dramatic. The most troublesome absence is probably *ptrace*, but it is hard to fault Tanenbaum for leaving it out, given his pedagogical intentions.

Another misconception is that Minix, at least in its present form, is a threat to the Unix† system. The system is a teaching toy, not a production system. For one thing, it doesn't swap: when a program wants to grow in size, it can only do so if there is physical memory available. On a small machine — Minix is written for the IBM PC — this is quite a limitation. There are other shortcomings too, such as the lack of any programmable editor; *ed*, *sed* and *awk* are all missing. And the list goes on. None of this is intended to belittle Tanenbaum's achievement, which is currently all the rage in the PC/Unix world, and was the main technical topic of discussion at the EUUG meeting (read on).

Someone does think Minix is a threat, though: Richard Stallman. His attempt to create *the* public domain system to compete with AT&T's Unix system has been undermined by Minix. Stallman even phoned Tanenbaum and chewed his ear off for his audacity in stepping on territory Stallman had claimed for himself. I wish the conversation had been recorded for us all to share.*

† Unix is a registered trademark of AT&T.

* Note added in proof, by Andy Tanenbaum: "Stallman never called me. He never even sent me e-mail. We communicate via one of his disciples (like the Dalai Lama) because, in the words of the disciple: 'Every minute Richard spends communicating means an additional 1 minute delay in the release of GNU.'"

EUUG on the Boat

Onward to Helsinki, to board the M/S Mariella, a huge floating resort hotel that housed the EUUG (European Unix User's Group) meeting. The Mariella leaves one of Helsinki or Stockholm at 18:00, arriving at the other city at 09:00 the following morning. Why a boat? Alcohol in Scandinavia is very expensive, and duty-free alcohol is essential to survival for some groups. These boats (there are several, owned by competing companies) make their money fairly equally from three things: transit fares, alcohol sold at the bars on board, and the duty-free stores on board. Once the boats leave port, the bars open and booze is cheap. Relatively cheap. Even by hotel standards, it was not cheap in my books.

Apparently these boats are a way of life in Finland. One Finn told me that, even though the return fare, including a single cabin, is only about \$80, the boats are a gold mine. They are busy all year round, and have a reputation for two things: cheap alcohol and 'casual physical relationships.'

There was considerable nervousness about the arrangements, and some active objections, both by the conference attendees and even by some of the organizers. Tanenbaum showed up in Stockholm to give his talk in port, refusing to go to sea in a 'floating liquor store.' The local Sun distributor in Helsinki wouldn't let any of its staff travel on the boat, for fear of tarnishing the company's reputation. (Do they know of Bill Joy's Ferrari?)

I think most people were surprised by how well it worked. It was a genuine novelty, although before long it just felt like a hotel, except you couldn't go out for take-out food. The boatiness of the beast was somewhat of a disappointment; I and others wanted tubes, tiny doors with wheels for handles and lots of roll. Instead, it was just a hotel. The only fun part was when it left Helsinki, during Dominic Dunlop's amusing, animated talk full of clever visual jokes. When the engines started, the slide projector began shaking, the floor began shaking, and the 6 inches or so of vibration in the image on the screen went well with the presentation.

The Joy of Superconduction

Bill Joy gave the keynote speech, and what a speech! The title was "Workstation architecture from 1982-1992," so I was prepared for lots of Bill's legendary log plots. I wasn't prepared for what he said, though.

First he spoke about a related topic: how to make money by selling workstations in 1982-1992. His trick is to make everyone happy by adhering to all possible protocol standards, and letting the power-of-two-every-five-years phenomenon give you enough horsepower to do so and enough log plots to keep your marketing speech current. Then the fun started.

Armed with the latest issues of Business Week and the Wall Street Journal, which he got in Frankfurt airport, he lectured us about superconductors. Ignoring completely the materials science aspect of things, he assumed that the superconductor industry would behave like the electronics industry in its exponential ways, and claimed that given high-temperature superconductors, Josephson junctions would finally achieve their long-promised ascendancy because their stumbling-block had been removed. He was obviously very excited, and to be fair, it made a hell of a good keynote speech, because his enthusiasm infected everyone.

The Mood

There's not much to say about the technical part of the EUUG meeting, except that it didn't quite work. There was a lot of after-hours talk in the on-board disco and casino about what was wrong. The consensus became roughly that most of the talks were about whether or not the Unix system was dead as a technical subject. If that's all you have to talk about, it's probably time to let the three-piecers in.

One certainty is that Tanenbaum's talk about Minix was the technical high point. It was also a very good talk. Everyone noticed that Minix is not commercial, and that in many ways Minix looks like the early Unix systems, not least in the availability of its source code. At universities and in the home hacker market, Minix may be the one to watch, although commercially it will be insignificant. One important difference, though, is that Minix will probably become more widely available — every student may have a private copy, instead of mere access to a university machine. That could make it develop in interesting ways. It is sure to be the basis for a number of innovations.

The Equipment Show

The vendor exhibition was necessarily small, because the conference section of the boat was much smaller than the liquor section. I was interested to see an Olivetti display (with give-away corkscrews) but no Unix Europe or AT&T display. Several book vendors were there, and Bjarne Stroustrup was horrified to discover his C++ book selling for \$80.

HP dominated the vendor show because they were making a big announcement, and wanted it to be backed up by a big display. As far as I can tell, they never said what their big announcement was to anyone but the press.

The Engines

I asked at the Information Desk on the Mariella if a small group of us could tour the engine room. No trouble, so at 23:00 down we went. Fortunately, one of our group spoke Swedish — despite what is claimed, English is not ubiquitous in Finland (more on that later).

It's hard to describe the noise, the sheer massiveness, of engines large enough to propel a luxury resort hotel through the Gulf of Bothnia. Here are some numbers that might help.

no. of engines:	4 V-12's, 9000 horsepower each, 36000 hp total
bore:	400 mm
stroke:	460 mm
power:	550 kW/cyl., 747 hp/cyl.
RPM:	455 in water, 520 in ice
mass:	66 tonnes
fuel:	137 gal/hph
mass:	
cylinder covers:	476 kg each
cylinder linings:	1000 kg each
pistons:	190 kg each
entire boat:	36000 tonnes

This strange mix of units is as the specifications were listed. The engines, and the entire boat, were manufactured by the Wärtsilä company in Helsinki (who were working on a Russian icebreaker when we toured Helsinki).

There are two screws 14.5 meters in diameter (or so we were told; this number contradicts the claimed draw of the ship). Direction of thrust is controlled by changing the pitch of the screws.

The figure about ice deserves comment. Finland leads the world in icebreaker technology. The Finnish government even defines standards for icebreakers. The Mariella is an icebreaker of the highest degree, and runs all year round.

Everything on the ship is monitored in the control room next to the engine room, using a couple of color displays with keyboards loaded with special function keys. All the controls are in English.

The 'user' interface was unimpressive at best.

Two men run the engines. When the boat is docked, 09:00-18:00 every day, local time, four more men are on board doing routine engine maintenance.

The nine-story boat draws only about six and a half meters.

We also went to the bridge. There is little to report — everything's pretty much as you'd expect, with computers monitoring the radar and so on (the ship was completed in 1985).

Something new to me: the stabilizers — little wings, one on each side of the ship — are driven relative to a gyro. On-board navigation uses radar and off-boat navigation aids, though.

Russian Computers in Finland

The Russians manufacture a number of computer clones that they sell in Finland, including a PC/AT, a VAX and a PDP-11. The PDP bus is different, though, for reasons I don't understand. But it's easy to deal with the different UNIBUS: you just need an adapter that inverts all the bus signals.

Two operating systems are available on the PDPs. One is the "Real Time" system that feels surprisingly like RSX-11/M, and the other is the "General Purpose" system with a striking similarity to the Seventh Edition Unix system (Tanenbaum beware).

One of Johan Helsingius's† friends spent a little while at a university in Moscow. One day he was stopped in the halls. "Pssst. Wanna buy a 4.2 source tape?"

The fact that the Russians have the Unix system helps Finnish companies that do business with Russia — and many do. Finland would not normally be permitted to sell Unix software to the Russians, but since they already have it anyway, it's not much of an issue in turnkey systems. Finnish computer salesmen must learn how to interpret the phrase, "does your product run on a PDP-11 under V7?"

The Russians also have a number of micros of various kinds. The Russian-manufactured 8086 still displays the Intel copyright notice if you pry open the case. Their 8085 didn't work properly, however, until an Intel employee visiting Finland was told about the problem. He recognized that the bug was one of the last bugs to be found during the development of the chip, and was still present in the last preproduction mask set. He deduced that the Russians stole the wrong masks. A few months later, Russian 8085's worked fine.

Language

Finnish is not an Indo-European language, it's a Finno-Ugric language. Nothing in common with English: 'Finnish' is 'suomi;' 'university' is 'yliopisto;' 'telephone' is 'puhelin.' The grammar is different: no future tense, no prepositions, no verb 'to have.' On the other hand, Finnish is perfectly phonetic (spelling is easy), has perfectly regular accents (always the first syllable), has no hard-to-say sounds and in fact relatively few sounds at all — 14 consonants, 8 vowels. So although it's supposed to be hard to learn, I decided to try a little. I was told before I left that everyone in Finland speaks English anyway; you don't need any Finnish. I believed English might get me by in Helsinki, but I figured the more Finnish I knew, the better I'd fare in Lapland. I was right. If nothing else, it's important to know the sentences, 'En puhu suomen' (I don't speak Finnish) and, 'Puhutekko englantia?' (Do you speak English?).

I did all right. In the hotels, people spoke enough English that I could get by. I only met one hotel employee who was genuinely fluent, and that was on my last night in a hotel. Elsewhere, especially outside Helsinki, English didn't accomplish much. I learned enough Finnish before I left, from a book Al Aho lent me, that I could understand menus and make myself understood from my vocabulary of a couple of hundred Finnish words, some rudimentary grammar, and a childhood fluency in grunts and gestures. By the end of the trip, I was getting by fairly well, particularly in restaurants. I negotiated most of my meals with no English, which is not to say I spoke perfect Finnish.

Finnish is such a non-European language that no foreigner learns it. The Finns aren't accustomed to hearing Finnish spoken badly, with a foreign accent. But they appreciate people trying, and beam from ear to ear when people do try. Rather than responding in English (if they could), they respond in slow simple Finnish. One restaurant owner was so impressed he refused to let Peter Langston, Ed Gould and myself pay for our Ranskalaiset (French fries). It was fun, and because I saw Finnish exclusively and heard Finnish almost exclusively (there was some English and Swedish and Russian on TV), it was starting to click into place after a couple of weeks, the way German did when I lived in Switzerland. I had to try harder to learn Finnish, though, no question about it.

I should mention that Finland is officially bilingual, the other language being Swedish, which is spoken as the first language by about 6% of the country, maybe 10% around Helsinki. This situation

† Johan was the conference organizer, and is a great guy to have around. Unfortunately, he's spending the next year in the Finnish army.

arose because the Swedes benevolently occupied Finland for a few hundred years, somewhat like the Normans in England. The Swedish-speaking population feels genuinely Finnish, though, not Swedish.

Television

In the north, the choice of channels goes down. Interestingly, the BBC, which is available in most of north-western Europe, is replaced by a Moscow channel in Lapland. The Sky channel, an English-language all-Europe channel, was everywhere. One night I had a choice between some Finnish documentary, a horrifically tasteless army brass band performing waltzes in a Moscow concert hall, and Hulk Hogan defending his wrestling title in Dayton on the Sky channel.

I began to see why people didn't pick up English from the TV.

Food

Russian food is the ethnic style to try in Helsinki. Other than that, things are pretty dull. The last night of the conference (only the technical sessions were on the boat; the tutorials were held on the Monday and Friday straddling the boat trip) I asked a hotel employee to recommend a restaurant. She recommended Suomelainen Ravintola, literally, Finnish Restaurant. It's the only one in Helsinki, probably the only one in Finland. Johan was horrified, but admitted he'd never been there so we forced him to come along.

He was glad he did.

The phrase 'Finnish cuisine' is interpreted to mean northern Finnish, with the characteristic elements such as reindeer and arctic cloudberry taken from Lapland. I won't attempt a food review here, but will mention a couple of surprises. The first was smoked elk meat, eaten raw, that dissolved superbly in the mouth (I had similar things at hotels in Lapland — they mostly offer 'Lappi a la carte' — but it was never as good as at Suomelainen Ravintola). The second were these arctic cloudbERRIES, bright orange and tart, used mostly as a meat garnish, but in some desserts. Their flavor was unusual, but they were ubiquitous, and would probably get dull after a while.

Of course, if there's only one Finnish restaurant, you have to find other places to eat on the road. Hotel food varied from mediocre to O.K., but was always expensive. The answer is to eat at a roadside 'grilli.' They're basically burger stands, but the cheeseburgers (hampurilainen juutosta) were very good, uniformly better than you could expect here. The french fries (ranskaleiset, or ranskasomething perunat) were good, too, if you spoke enough Finnish to prevent them being drowned in relish (kuukosomething salaati) and ketchup. My Finnish was just good enough, and my gestures helped.

The Finns drink a lot of coffee. More on that in a moment.

Architecture

One of the reasons I wanted to visit Finland was the architecture there. It's a long story, but permit me to tell a compressed version. Finland has produced a number of great architects — IBM Yorktown and Holmdel were designed by one of them, Saarinen (Junior) — but one architect that worked almost exclusively in Finland, and central Finland at that, and who is regarded reverently by the Finns. His face appears on the new fifty-mark notes. His name is Alvar Aalto. My interest in architecture can be traced entirely to a retrospective of his at the New York Museum of Modern Art a few years ago, and he designed an astonishingly large fraction of Finland.

Some towns, such as Otaniemi (where the tutorials were held) and Jyväskylä, have almost all the major buildings done by Aalto. There is much to say about Aalto's style, and I am hardly the person to do him justice, but I took some of the EUUG attendees on an architectural tour of Otaniemi and they agreed with me that his work was special. Although his buildings are not particularly exciting from the outside, his interior spaces are amazing, open, full of light and inviting. He also designed beautiful, simple, functional and very comfortable furniture.

Even the buildings Aalto didn't design are interesting. The northern part of the country was almost literally burnt to the ground by the retreating Germans at the end of the Second World War, and a massive reconstruction program began to house the displaced inhabitants. With materials scarce, it

was not feasible to build many single dwellings, so the towns are full of apartments and town houses, which sounds awful. But they are very successful, blending with the landscape and not at all obtrusive to the eye. Aalto's influence (he came to prominence in the 1920's and worked until he died, in 1976) is evident everywhere; his use of light and space and wood dominates the style of buildings. The hotel rooms, for instance, are spare but airy, bright and cozy. Without implying any stylistic similarity, the Finnish buildings reminded me of the Japanese control of space.

The Last 24 Hours

My last day in Finland was remarkable enough to be recorded chronologically. I had driven from Inari the day before, through Pelkosenniemi where the army was trying to clear the ice floes from the streets where they had floated when the army dynamited the ice dam in the river, and spent the night at a hotel/hotel school in Rovaniemi. I spent the day in Rovaniemi, mostly at the Aalto-designed library, and in the evening drove to the train station.

The Finns have a great idea in trains: an overnight sleeper train runs between Helsinki and Rovaniemi, and for about \$100 you get a bunk, breakfast and a place on the train for your car. So you go to sleep at one end of the country and wake up at the other end with your automobile, for not much more than the cost of a hotel room for a night.

My companion in the sleeper compartment was a Lapp, whose native language is Saami, but who spoke Finnish and, of course, Swedish. Nonetheless, we had a great time for a couple of hours passing my Berlitz back and forth. You really can have a conversation that way.

The next morning, in Helsinki, I tried to find the Artek store. Artek is the furniture company founded by Alvar Aalto, and their store is right in the heart of the Helsinki shopping district. I found a building labeled Artek, but the store there looked like a Marimekko. So I tried another entrance and found a pair of elevators and a building directory: Artek, 8th floor. Up I went.

There was a receptionist, so I asked her, "Puhutekko englantia?" She panicked, and took me to the back to look for someone else. Sure enough, we found some English speakers: two architects in a studio with beautiful skylights. I explained that I was looking for the Artek store, and that I was an architectural pilgrim and a fan of Aalto. They were delighted. They gave me a catalog and the name of an American distributor of Artek furniture, and gave me the grand tour. Martti (I have, ashamedly, forgotten his last name) was in the process of designing furniture while I was there, but took time out to show me around and explain some things. Of course, the building itself was designed by Aalto (hence the skylights). Martti was young and enthusiastic, and took me to the ground floor and showed me the Artek store. It was, indeed, hidden somewhat. First, though, he showed me Galleria Artek, which was displaying some work by 'Carpenter Kari Virtanen.' These were new pieces of furniture, hand made and insanely expensive, but simply beautiful, and beautifully simple. They were made of birch, nothing else, with all hidden-joint construction and imaginative laminations. For example, the front doors of one cabinet were laminated with the strips running vertically, about 3mm wide, with colored glue, so the doors looked like they were vertically ruled.

The Artek showroom, at last, had all the Aalto furniture there for me to try out. Even better, though, Martti explained the innovations in the various pieces, and gave me a personal lecture on modern Scandinavian furniture design. I was not surprised, but I was dismayed, to learn that Finnish furniture is remarkably inexpensive in Finland. It's not here.

Next, to the airport, where I spent half an hour finding where to leave my well-driven 1.1 litre Ford Fiesta. On the flight to Amsterdam I sat beside a Dutch gentleman.

"Are you a Dutchman?" he asked me, in English.

"No," I said, "I'm Canadian."

"Then we can talk in English." And we did. He was in the lumber business, and spent a lot of time in Finland. His language of commerce was Swedish. About this time the stewardess came by with drinks.

"Would you like some coffee?" she asked him, in English.

"No, thanks. Coffee, coffee, coffee, I'm sick of coffee. I spend all day driving from sawmill to sawmill and every time I stop it's coffee, coffee, coffee. I am full of coffee."

The stewardess was taken aback.

;login:

The USENIX Association Newsletter

Volume 12, Number 4

July/August 1987

CONTENTS

Call for Papers: POSIX Portability Workshop	3
Call for Papers: Winter 1988 USENIX Conference	4
Call for Papers: Summer 1988 USENIX Conference	5
Computer Graphics Workshop	6
Multiple Programs in One UNIX Process	7
<i>Don Libes</i>	
tar vs. cpio	13
How To Write a UNIX Daemon	17
<i>Dave Lennert</i>	
Call for Papers: EUUG Spring 1988 Conference	24
Book Reviews	25
The Design of the UNIX Operating System	25
<i>Marc D. Donner</i>	
A C Reference Manual (Second Edition)	27
<i>Josiah C. Hoskins</i>	
UUNET Progress Report	28
The 1988 Election of the USENIX Board of Directors	30
Call for Participation: USENIX C++ Workshop	30
Summary of the Board of Directors' Minutes New Orleans, 26-27 March 1987	31
Summary of the Board of Directors' Meeting Phoenix, June 7, 8, 10, 1987	32
Future Meetings	34
Publications Available	35
4.3BSD UNIX Manuals	36
4.3BSD Manual Reproduction Authorization and Order Form	37
Local User Groups	38

The closing date for submissions for the next issue of *;login:* is August 28, 1987



:login:

Call for Papers POSIX Portability Workshop

Berkeley Marina Marriott

October 22-23, 1987

This USENIX workshop will bring together system and application implementors faced with the problems, "challenges," and other considerations that arise from attempting to make their products compliant with IEEE Standard 1003.

The first day of the workshop will consist of presentations of brief position papers describing experiences, dilemmas, and solutions. On the second day it is planned to form smaller focus groups to brainstorm additional solutions, dig deeper into specific areas, and attempt to forge common approaches to some of the dilemmas.

Suggestions for topic areas and position papers include:

C Language Issues	Internationalization
Networked/Distributed Implementations	Pipes and FIFOs
Timer resolution, ranges	Signals
Conformance verification	Security concerns
Job control, process groups	Limits: documentation and inquiry
Implications for user interfaces	Implications for commands

Position papers must be submitted by August 15, 1987 to:

Jim McGinniss
Digital Equipment Corporation
Continental Boulevard MK02-1/HIO
Merrimack, NH 03054
(603) 884-5703
decvax!jmcg
jmcg@decvax.DEC.COM

For registration or hotel information, contact:

Judith F. DesHarnais
USENIX Conference Coordinator
PO Box 385
Sunset Beach, CA 90742
(213) 592-3243
usenix!judy

:login:

Call for Papers Winter 1988 USENIX Conference

Dallas, Texas
February 9-12, 1988

Please consider submitting an abstract for your paper to be presented at the Winter 1988 USENIX conference. Abstracts should be around 250-750 words long and should emphasize what is new and interesting about the work. The final typeset paper should be 8-12 pages long.

The Winter conference will be four days long: two days of tutorials only and two days of papers only.

Suggested topic areas for this conference include (but are not limited to):

- Electronic Publishing
- Novel Kernels
- New Software Tools
- New Applications
- System Administration
 - (including distributed systems and integrated environments)
- Security in UNIX
- Future Trends in UNIX

This conference may include a "miscellaneous" session which will include those papers which normally do not fit into normal tracks. Vendor presentations should contain technical information and be of interest to the general community.

Abstracts are due by **October 23, 1987**; papers absolutely must be submitted by **January 4, 1988**. Notifications of acceptance of abstracts will be sent out by November 6. Papers that do not meet the promise of their abstract will be rejected. Talks will be given on all papers published in the *Proceedings*; failure to submit a paper for an abstract will result in forfeiture of the talk.

►Please contact the program chairman for additional information:

Rob Kolstad
CONVEX Computer Corporation
701 Plano Road
Richardson, TX 75081
214-952-0351 (W)
214-690-1297 (H)
214-952-0560 (FAX)
{usenix,ihnp4,uiucdes,allegra,sun}!convex!kolstad

Please include your network address (if available) with all correspondence. It should be an ARPANET (EDUNET, COMNET), BITNET, or CSNET address or a UUCP address relative to a well-known host (e.g., *mcvax*, *ucbvax*, *decvax*, or, *ihnp4*).

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

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

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

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

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

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

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

Send technical program submissions to:

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

;login:

Computer Graphics Workshop

Boston Marriott Cambridge
Cambridge, MA
October 8-9, 1987

The Fourth USENIX Computer Graphics Workshop will be held at the Boston Marriott Cambridge in Cambridge, MA, October 8 and 9, 1987, with a no-host reception on the evening of October 7.

Registration will be \$200 per attendee and must be paid in advance. There will be no on-site registration.

There is a special hotel rate for workshop attendees of \$115 per night, single or double. Call the Marriott direct for reservations: 617-494-6600. Be sure to mention that you are a USENIX Workshop attendee. The Marriott has a strict cut-off of September 16 for its special rate. Reservations made after that date will be on a space and rate available basis.

Partial Program

The BRL CAD Package – Michael John Muuss & Phillip Dykstra (BRL)

REMRT – A Network Distributed and Parallel Ray-Tracer – Michael John Muuss (BRL)

The Definition and Ray-tracing of B-Spline Objects in a Combinatorial Solid Geometry Modeling System – Paul R. Stay (BRL)

More Music Software for UNIX – Michael Hawley (MIT)

Dynamics for Everyone – Jane Wilhelms (UCSC)

Distributed Computation for Computer Animation – John W. Peterson (Utah)

It's all done with Smoke and Mirrors – The Face Saver Project –
David Yost & Lou Katz (Consultants)

Paint Systems and Images of Arbitrary Size and Shape –
Ken Knowlton & Lou Katz (Consultants)

Hairy Brushes – Steve Strassman (MIT)

= = =

For further program information, contact:

Tom Duff at *research!td* or Lou Katz at *ucbvax!lou*.

For registration information, contact:

Judith F. DesHarnais
USENIX Conference Office
P.O. Box 385
Sunset Beach, CA 90742
(213) 592-3243
usenix!judy

NOTE: Make your hotel reservations on or before September 16!

Multiple Programs in One UNIX Process

Don Libes

National Bureau of Standards
Bldg 220, Rm A-127
Gaithersburg, MD 20899

(301) 975-3535
(seismo,umcp-cs)!nbs-amrf!libes

ABSTRACT

A small operating system (XINU) was ported to UNIX 4.2BSD. The entire operating system runs as a single UNIX process. The code is approximately 1000 lines of C (including comments) and 6 lines of assembler.

All of the code is user-level, and thus presents a system easy to examine, understand, and experiment with further.

The code has been used as a base for an application of several cooperating processes communicating through global variables. Alternatively, the system provides semaphores and messages for interprocess communication.

Background – Why Did We Need This?

This project fell out of a recent porting effort at NBS. The original desire was to move an application from a non-UNIX computer to a UNIX computer. The non-UNIX computer ran a simple home-brewed operating system the details of which are unimportant except that it provided interprocess communication through global variables. While 4.2 promised shared memory, it failed to deliver on this. (This has since been remedied by [Libes 85].) To quote from the manual page for `mmap(2)` [Joy 83]:

DESCRIPTION

N.B.: This call is not completely implemented in 4.2.

Taking the cryptic advice, we decided that it might be possible to port the entire operating system and application as a single UNIX process.

This proved to be possible with the help of several recent enhancements of 4.2 UNIX including sub-second interval timers and non-blocking I/O. Our first implementation did not require separate process stacks, and we realized that by adding them, we would have a tool of much more generality. Before proceeding much further, we quickly realized the similarity to XINU as presented by [Comer 84]. (Other approaches are discussed by [Kepecs 85] and [Tevanian 87].)

XINU

In *Operating System Design, The XINU Approach* Douglas Comer presents a layered and modular operating system. In contrast to other operating system texts which compare and contrast a variety of algorithms for typically only the most interesting tasks in an operating system, Comer chooses one technique for each problem, usually the most straightforward one or that leading to the simplest presentation. (Alternatives are often proposed in the exercises at the end of each chapter.)

The book is further unique in discussing every last aspect of a single implementation. This implementation is XINU. The entire source of XINU is in the text, including the machine dependent code for running XINU on a Digital Equipment Corporation LSI 11/2 (a microcomputer version of the PDP 11).

:login:

Based on Comer's unusually well-written text, we felt that it might be possible to bring up XINU on top of UNIX. Such a system would be able to provide the concurrency and shared variables that our original application needed, and at the same time be immediately useful to others, since it was already well-documented.

In fact, it was not hard. Our task was much easier than Comer's in that we had a complete set of device drivers already. This included a file system and terminal interface. We also did not have to worry (much) about operating system startup and C start up.

It took approximately 3 hours to type in the necessary parts of XINU. This included process management and utilities for fifo and priority queues. Our initial version of XINU did not use a real-time clock. Processes had to explicitly give up control (through calls to `wait`, `sleep`, etc). During that time, it was possible to use `setjmp/longjmp` to switch between processes. `setjmp/longjmp` saves/restores the registers including the `pc` (program counter) and `sp` (stack pointer), and also the signal mask (used as an interrupt mask).

Process Rescheduling – `resched()`

XINU processes call `resched` (via `wait`, `sleep`, etc) to give up control temporarily of the processor to a ready process. Here is the code fragment where an old process gives control to a new process.

```
/* _resched.c - reschedule and context switch processes
_resched() {
    .
    .
    /* old process is running */
    if (0 == setjmp(optr->pregs)) longjmp(nptr->pregs,1);
    /* new process resumes here */
    .
    .
}
```

Each process has a structure describing its state whenever the process is not currently executing (analogous to the `_u` structure in the UNIX kernel). In the above code fragment, `nptr` points to the new process to begin executing. `optr` points to the old process that is going to be suspended. The field `pregs` is the register save area. All that is necessary to switch processes is to save the current register values in `optr`, and restore the old register values in `nptr`.

`setjmp` saves most of the registers including the `pc` and `sp`. However, it does not save the condition codes. This is because `setjmp` and `longjmp` are never immediately followed by a test of the condition codes.

Using the 4.2BSD interval timer, we added a real-time clock. The real-time clock could interrupt computations anywhere, including the case where the conditions code had been set but not tested. At this point, it became necessary to do context switches ourselves. That required a small number of assembler statements.

The following amended version of `resched` (for an MC68000) can be called from interrupt handlers as well as user processes.

```
/* _resched.c - reschedule and context switch processes */
/* Since we can't pass parameters to rte from resched, we use these */
/* variables that are global to both routines. */
static int *new_sp;          /* new stack pointer register to be loaded */
static int (*new_pc)();       /* new program counter register to be loaded */
static int new_signal_mask;   /* new signal mask to be loaded */
void _resched()
{
    register struct pentry *optr; /* pointer to old process entry */
    register struct pentry *nptr; /* pointer to new process entry */
```

;login:

```
/* no switch needed if current process priority higher than next */
if (((optr = &_proctab[_currpid])->pstate == PRCURR) &&
    (lastkey(_rdytail)<optr->pprio))
    return;
/* if the old process was still runnable, mark READY */
if (optr->pstate == PRCURR) {
    optr->pstate = PRREADY;
    _insert(_currpid,_rdyhead,optr->pprio);
}
/* remove highest priority process at end of ready list */
nptr = &_proctab[(_currpid = _getlast(_rdytail))];
nptr->pstate = PRCURR; /* mark it currently running */
#ifndef RTCLOCK
    /* schedule an interrupt for the end of a quantum or the next event */
    /* in the sleep queue, whichever is sooner */
    _start_itimer((*_slnempty && (*_sltop < QUANTUM))?*_sltop:QUANTUM);
#endif
/* ctxsw(optr->pregs,nptr->pregs);*/
/* at this point, optr->pregs == a5@, nptr->pregs = a4@ */
/* save all registers in optr->pregs */
asm("moveml #0xffff,a5@"); /* save all the registers */
asm("movl #OLDPROC,a5@(64)"); /* change pc and save it */
optr->signal_mask = sigblock(0); /* save old interrupt reg */
/* we have completed putting the old process to bed */
/* now restart the new process */
/* prepare pc, sp and interrupt mask for rte() to use */
new_sp = nptr->sp; /* movl a4@(60),_new_sp */
new_pc = nptr->pc; /* movl a4@(64),_new_pc */
new_signal_mask = nptr->signal_mask;
/* load rest of registers directly except for a7 (sp) */
asm("moveml a4@,#0xffff"); /* restore d0-d7,a0-a3 */
asm("moveml a4@(52),#0x6000"); /* restore a5-a6 */
asm("movl a4@(48),a4"); /* restore a4 */
kill(getpid(),RTE);
fprintf(stderr,"resched: kill(),RTE) returned?0);
/* old process returns here */
asm("OLDPROC:");
}
```

Notice that the scheduler here is very simplistic. Highest priority processes are selected round-robin. (More complex schedulers might use per-process quanta as well as reassigning priorities.)

```
/* if the old process was still runnable, mark READY */
if (optr->pstate == PRCURR) {
    optr->pstate = PRREADY;
    _insert(_currpid,_rdyhead,optr->pprio);
}
/* remove highest priority process at end of ready list */
nptr = &_proctab[(_currpid = _getlast(_rdytail))];
```

An interval timer is then scheduled to occur at the end of the next quantum or for the first scheduled sleeping process, whichever is sooner.

```
/* schedule an interrupt for the end of a quantum or the next event */
/* in the sleep queue, whichever is sooner */
_start_itimer((*_slnempty && (*_sltop < QUANTUM))?*_sltop:QUANTUM);
```

The real-time clock is simulated using the 4.2 interval timer. Rather than generating constant interval clock ticks, the interval timer is only set for known events (i.e. quanta and sleeping processes). This reduces the number of clock interrupts significantly.

Context Switching – rte()

Comer describes (p. 59) the LSI instruction `rtt` (return from trap) which reloads the `pc` and `ps` (processor status register) at the same time. We have a similar problem, although rather than reloading the `ps`, we want to reload the signal mask, `sc_mask`. The solution is to artificially provoke a signal (via `kill`) which at termination executes a `rte` (return from exception). This is the 68000's analog to the 11/2's `rtt`.

```
/* rte() - indirectly execute 68K rte (return from exception) instruction */
/* Always called from resched().  This routine is necessary to load the */
/* signal mask at the same time as we load the new pc and sp.  */
/* setjmp/longjmp is unusable as it doesn't save/restore all the registers. */
/* ARGSUSED */
static void rte(sig,code,scp)
int sig;
int code;
struct sigcontext *scp;
{
    scp->sc_sp = (int)new_sp;
    scp->sc_pc = (int)new_pc;
    scp->sc_mask = new_signal_mask;
    /* No need to reload ps, as no one looks at it anyway, upon return. */
```

A Simple Example

We want two XINU processes to execute simultaneously, one continuously printing "1", and the other continuously printing "2". To do it, we create two subroutines as follows:

```
prog1()
{
    for (;;) printf("1");
}
prog2()
{
    for (;;) printf("2");
}
```

The following subroutine is all that is necessary to run them.

```
user_main()
{
    xresume(xcreate(prog1,2000,20,"prog1",0));
    xresume(xcreate(prog2,2000,20,"prog2",0));
}
```

Compiling this together with the XINU support routines and running the executable produces the following output:

```
1111122222111122222111122222111122222 . . .
```

`xcreate` takes a subroutine and creates a runnable (XINU) process, returning a process id. Passing the process id to `xresume` allows the process to run. The remaining parameters to `xcreate` are the stack size, a process priority, a tag for debugging, and a number and list of arguments passed to the process when started. Further information can be found in Comer's book.

Miscellaneous But Important Implementation Notes

The entire project took approximately two person-weeks. This included typing in the source, learning the necessary amount of both LSI 11/2 assembler (Comer's original) and a mongrel 68K/UNIX assembler provided by the vendor of our 4.2 system. Lastly, we had to figure out the undocumented C calling conventions for the 4.2 C compiler (very similar to what Comer discusses) as well as experiment with the undocumented `asm` statement in our C compiler.

:login:

Although we have no references on it, `asm` is a keyword in (apparently) many C compilers which allows the user to drop assembler statements into the C compiler's assembler output. For example,

```
foo();  
asm("jsr bar"); /* bar(); */
```

calls `bar` after calling `foo`. The next logical step doesn't work,

```
sprintf(asm_buffer,"jsr %s","bar");  
asm(asm_buffer);
```

evokes the error *syntax error at or near "asm_buffer"* from the 4.2 C compiler. You should try this on your particular C compiler.

4.2 XINU System Calls

The supported system calls are:

<code>xsend()</code>	send a message to another process
<code>xreceive()</code>	wait for a message and return it
<code>xrecvclr()</code>	clear messages, returning waiting message (if any)
<code>xresume()</code>	unsuspend a process, making it ready
<code>xsuspend()</code>	suspend a process, placing it in hibernation
<code>xkill()</code>	kill a process and remove it from the system
<code>xcreate()</code>	create a process to start running a new procedure
<code>xgetpid()</code>	get the process id of currently executing process
<code>xgetprio()</code>	return the scheduling priority of a given process
<code>xchprio()</code>	change the scheduling priority of a process
<code>xwait()</code>	make current process wait on a semaphore
<code>xsignal()</code>	signal a semaphore, releasing one waiting process
<code>xscreate()</code>	create and initialize a semaphore, returning its id
<code>xsdelete()</code>	delete a semaphore by releasing its table entry
<code>xsleep()</code>	put a process to sleep for this many seconds
<code>xmsleep()</code>	put a process to sleep for this many milliseconds

For complete documentation on the system calls, see Comer's text. Most of the supported system calls function exactly as described in the book. The only changes were to provide a millisecond timer rather than a tenth of a second timer, and all XINU system calls are prefaced with 'x' (for XINU) to avoid clashes with UNIX calls.

All internal procedures and variables that are global have been prefaced with an underscore to avoid conflicting with application names. For example, `_resched`.

The system is configurable and can be recompiled without any combination of the following optional services:

- realtime clock
- semaphores
- messages

These and other typical configuration changes are isolated in `_conf.h`.

The smallest 4.2 XINU system comes with only 5 system calls:

```
xcreate()  
xresume()  
xsuspend()  
xkill()  
xgetpid()
```

and is actually quite useful.

:login:

Other Minor Differences Between Comer's XINU and 4.2 XINU

Comer's XINU is based on the LSI 11/2. 4.2 XINU is based on 4.2BSD UNIX. The source is almost entirely in C, and makes few assumptions about the underlying machine. Much of the code ports without changes. Besides the differences mentioned elsewhere in this paper, the other primary differences are the size of the registers and interrupt handling.

The LSI is 16-bit while 4.2 is `char *`. Occasional assumptions are necessarily made, unfortunately.

Interrupts in the UNIX appear as software signals. Thus, disabling interrupts is done with the 4.2 signal support routines.

If you intend to write your own system calls, you must allocate an `int` to store the old interrupt mask rather than a `char`. For example, `disable(ps)` is used to disable interrupts while storing the old processor status in `ps`. Comer's definition of `disable` is:

```
/* disable interrupts - LSI 11/2 */
#define disable(ps)      asm("mfps ~ps"); asm("mtps $0340")
```

while for 4.2 XINU, it is

```
/* disable interrupts - 4.2BSD software interval timer */
#define disable(oldmask)    oldmask = sigblock(1<<(SIGALRM-1))
```

Here, only the quantum timer interrupt is blocked. You may find that other signals should be blocked, however not all should (e.g. SIGTSTP should probably not be trapped).

Using UNIX System Calls From XINU

All UNIX system calls and many library calls should be made with some thought as to their consequences. `exec`, for example, will completely overlay the entire XINU application and system. Where functionality is duplicated by UNIX, it is generally better to use XINU's calls. For example, if a process wants to go to sleep, calling the UNIX `sleep` will stall the entire XINU system until the next interval timer occurs. If the process calls `xsleep` (the XINU equivalent) the current process is put on a queue waiting for the clock, and another XINU process is given control of the cpu.

We have not reimplemented I/O, since we were able to use UNIX I/O without change, however the default behavior of UNIX I/O is to block, leading to a similar problem as `sleep` (i.e. block until operation complete or until quantum expires).

Note that 4.2 system calls restart automatically upon interrupts. This allows programs to run without having to explicitly handle the quantum interrupt.

If this "blocking until quantum" behavior is undesirable, it is possible to use non-blocking I/O, either directly, or through a generalized interface leading to a second set of I/O system calls. Future work in this direction would be very useful.

Using UNIX Library Calls From XINU

Many UNIX library calls are nonreentrant, and do not protect themselves against this. This means that they use static variables which are common from one call to the next. If two processes make the same nonreentrant library call at the same time, it is likely that the routines will misbehave.

Using reentrant versions of libraries is the best solution. Alternatively, one can embed (or surround, if you don't have the source) semaphores in the library calls (provided by XINU), one per common data structure (such as `_iob` which is shared with all the routines that are part of the standard I/O library).

XINU system calls are protected against reentrancy problems by disabling timer interrupts. Since `malloc` and `free` are used for XINU memory management, you should disable timer interrupts or set up a semaphore for access to the `malloc` data structures when doing memory management.

:login:

Conclusion

We have ported an operating system to the UNIX environment by emulating the environment of a microprocessor in a single UNIX process. We now have a tool that is capable of simulating any set of cooperating real-time processes.

The applications have the ability to access all the power of UNIX, simply because the emulator runs as normal user code on a 4.2BSD system. Because all the XINU processes run in one UNIX process, it is especially easy to debug multiple programs with one debugger.

Perhaps the nicest benefit of this work, has been the ability to write processes that efficiently share data structures, at the expense of using distinct global names. This has long been a missing feature of UNIX.

References

- Comer, Douglas. *Operating System Design, The XINU Approach*. Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1984.
- Joy, W., Cooper, E., Fabry, R., Leffler, S., McKusick, K., Mosher, D., "4.2BSD System Interface Overview," Computer Systems Research Group, U.C. Berkeley, July, 1983.
- Libes, Don. "User-Level Shared Variables." Tenth USENIX Conference Proceedings, Summer 1985.
- Kepecs, Jonathan. "Lightweight Processes for UNIX Implementation and Applications." Tenth USENIX Conference Proceedings, Summer 1985.
- Tevanian, Jr., A., Rashid, R., Golub, D., Black, D., Cooper, E., Young, M., "Mach Threads and the UNIX Kernel: The Battle for Control." Summer 1987 USENIX Conference Proceedings.

= = =

tar vs. cpio

The following memorandum was delivered at the June meeting of the P1003 committee in Seattle by John S. Quarterman, USENIX Institutional Representative to the Committee. I feel its content is of great importance to the membership and have reproduced it here with John's consent. The final note is a consequence of the June meeting. - PHS

Secretary, IEEE Standards Board
Attention: P1003 Working Group
345 East 47th St.
New York, NY 10017

In both the Trial Use Standard and the current Draft 10, POSIX §10.1 describes a data interchange format based on the tar program. That section has appeared in every draft of IEEE 1003.1 in some form and has always been based on tar format. The P1003.1 Working Group has recently received two related proposals regarding that section: one to add cpio format (including old-style, non-ASCII

(non c option) format); [N.048 Lorraine C. Kevra] [V11N14] [V11N25 Eric S. Raymond] the other to replace the existing tar-based format with cpio format. [N.043 X/OPEN] [V11N13] Some clarifications were received to the former. [N.064 Dominic Dunlop] [V11N15] It was also proposed verbally in the latest Working Group meeting to drop §10.1 altogether and let P1003.2 handle the issue. [V11N08] [V11N11] [V11N09 Guy Harris] [V11N12 Doug Gwyn]

The present note is a response to those proposals. Much of the detail in it is derived from articles posted in the USENET newsgroup

`comp.std.unix`. Those articles are referenced with this format: [V11N09 Guy Harris] which gives the volume (always 11) and number of the article, and the name of the submitter. If no submitter name is given, the posting was by the moderator, John S. Quarterman. Thanks to those who submitted articles. However, the content of this note is solely the responsibility of the author.

This note is addressed to P1003.1, and is concerned with data interchange formats. Although user interface issues may be of interest to P1003.2, they are not addressed here.

There are a number of problems with both cpio formats. First, those related to the non-ASCII format:

1. Numerous parameters, including inode numbers, mode bits, and user and group IDs, are kept in two-byte binary integers. This has historically produced serious byte-order problems when data is moved among systems with different byte orders. [V11N09 Guy Harris]

2. The byte-swapping and word-swapping options to the cpio program are inadequate patches; with an ASCII format the problem would not be present. The options are not consistent across versions of the program: in System III, data blocks and file names are byte swapped; in System V, only data blocks are byte swapped. [V11N09 Guy Harris] [V11N47 Andrew Tannenbaum]

3. The two-byte integer format limits the range of inode numbers to 0..65535. Many current file systems are bigger than that. [V11N37 Paul Eggert] [V11N39 Henry Spencer]

Non-ASCII cpio format is clearly not portable and should not even be considered for standardization. [V11N12 Doug Gwyn]

There are several problems that occur even with the ASCII cpio format:

1. Many implementations of cpio only look at the lower 16 (or even 15) bits of the inode number, even in ASCII format. [V11N39 Henry Spencer] This is because the variable that is used to contain the value is declared to be unsigned short, just as in binary format. Thus, even though ASCII cpio format only

constrains this number to the range 0..262143, the format is still less than portable. [V11N37 Paul Eggert]

2. The proposed cpio ASCII format as specified, [N.048 Lorraine C. Kevra] [V11N14] is not portable because the proposal assumes that `sizeof(int) == sizeof(long)`. [N.064 Dominic Dunlop] [V11N15]

3. The file type is written in a numerical format, making it UNIX specific rather than POSIX specific, since POSIX (and tar) specifies symbolic, rather than numerical, values for file types. [V11N09 Guy Harris]

4. Hard links are not handled well, since cpio format does not directly record that two files are linked. If two files that are linked are written in cpio format, two copies will be written. The cpio program detects duplicate files by matching pairs of (`h_dev`, `h_ino`) and producing links, but that is done after the fact. [V11N09 Guy Harris] [V11N45 Guy Harris] [V11N54 Ian Donaldson] (There is a program, afio, that handles cpio format more efficiently in this and other cases than the licensed versions of the program.) [V11N21 Chuck Forsberg]

5. Symbolic links are not handled at all, and no type value is reserved for them. This makes cpio useless on a large class of historical implementations (those based on 4.2BSD or its file system) for one of the main purposes of POSIX §10.1: archiving files for later retrieval and use on the same system. Although it is possible to extend cpio to handle symbolic links, and at least one vendor has done this, [V11N45 Guy Harris] the format proposed to P1003.1 is the format in the SVID, and does not handle symbolic links.

6. The cpio format is less common than tar format: there are few historical implementations from Version 7 on that do not have tar; there are many that do not have cpio. [V11N09 Guy Harris] [V11N10 Charles Hedrick] [V11N24 Jim Cottrell] It is true that cpio (non-ASCII format) was invented before tar, [V11N22 Joseph S. D. Yao] apparently in PWB System 1.0. [V11N26 Joseph S. D. Yao] The cpio program was first available outside AT&T with PWB/UNIX 1.0, [V11N45 Guy Harris] [V11N63 Joseph S. D. Yao] and later with System III. However, in the interim, Version 7, which did not include cpio [V11N53 Bill Jones] [V11N62 Guy Harris] but

did include tar, became the most influential system. There was a V7 addendum tape, but it also did not include cpio (according to its README file); [V11N65 Rick Adams] the addendum tape was in tar format. Also, it appears that the cpio format of PWB was not the same as that of System III. [V11N39 Henry Spencer] And System III and all releases of System V include tar. [V11N26 Joseph S. D. Yao] [V11N63 Joseph S. D. Yao] [V11N45 Guy Harris] [V11N47 Andrew Tannenbaum]

7. It is very late in the process to propose that P1003.1 adopt cpio format now, especially considering that it was originally proposed to and rejected by the /usr/group committee before P1003.1 was even formed. [V11N39 Henry Spencer]

Advantages of cpio format include:

1. Both X/OPEN [N.043 X/OPEN] [V11N13] and the SVID [N.048 Lorraine C. Kevra] [V11N14] use it, although evidently defined somewhat differently. [N.064 Dominic Dunlop] [V11N15]

2. Archives made in cpio format are often smaller than ones in tar format. [V11N44 Mark Horton] But this is only because of the headers, and thus the effect diminishes with larger files.

3. On a local (non-networked) system, cpio is more efficient at copying directory trees than tar. [V11N46 Steve Blasingame] However, this is really an implementation issue.

There are several advantages to the current tar-based format as specified in §10.1:

1. There are no byte- or word-swapping issues caused by the format, since all the header values are ASCII byte streams. [V11N17 John Gilmore]

2. There are no inode numbers recorded, and file types are kept in symbolic form, so the format is less implementation-specific than cpio format. [V11N17 John Gilmore]

3. Historical tar format is the most widely used, as discussed in 6. above, despite apparent assertions to the contrary. [N.043 X/OPEN] [V11N13]

4. The format specified in §10.1 is upward-compatible with tar format. Old tar archives

can be extracted by a program that implements §10.1. Archives using some of the extensions of §10.1 can be extracted with old (Version 7) tar programs, although symbolic links will not be extracted and contiguous files will not be handled properly (cpio does not handle these capabilities at all). Files with very long names will not be handled properly (cpio does no better at this). All tar implementations are compatible to this extent. [V11N17 John Gilmore]

5. The /usr/group working group and P1003.1 have already done the work [P.061] [M.019 5.1.121 Pg.13] [RFC.003 #121] [P.038] [P.006] required to add optional extensions (such as symbolic links, long file names, [V11N49 Jerry Schwarz] [V11N50 Michael MacDonald] and contiguous files) that are needed on many historical implementations and that cpio format lacks.

6. The format is extensible for future facilities. [V11N39 Henry Spencer]

7. There is a public domain implementation of the format of §10.1. That implementation provided feedback which led to improvements in the current specification, and has been in use for years in transferring data with licensed tar implementations. [V11N17 John Gilmore]

8. Many people prefer the user interface of the cpio program to that of the tar program, because the former can accept a list of pathnames to archive on standard input while the latter takes them as arguments, limiting the length of the list. [V11N34 Andrew Tannenbaum] However, the above-mentioned public domain implementation of tar accepts pathnames on standard input, [V11N17 John Gilmore] [V11N19 Jim Cottrell] and at least one vendor sells a version of tar that can do this. [V11N48 Michael Gersten] Diffs to standard tar to add an option to accept pathnames on standard input when creating an archive have also been posted to USENET. [V11N36 John Gilmore] The user interface is, in any case, irrelevant to P1003.1. [V11N39 Henry Spencer] [V11N40 Rahul Dhesi]

Disadvantages of tar format:

1. If an attempt is made to extract only the second of a pair of hard linked files the tar program will attempt to link the second file to the nonexistent first file, and nothing will be

extracted. Although a sufficiently clever implementation could avoid this, the problem can be considered to be in the archive format. [V11N66 Kenneth Almquist]

There are some problems that neither tar nor cpio handles well.

1. File names still longer than the length of PATH_MAX (at least 255) [V11N50 Michael MacDonald] that the POSIX format allows (and than the 128 that cpio permits or than the 100 that historical tar allows) would be preferable, although the POSIX limit is useful for most cases. [V11N54 Ian Donaldson]

2. An option to prevent crossing mount points would be useful for backups. [V11N19 Jim Cottrell] [V11N22 Joseph S. D. Yao] However, this appears to be more of an implementation issue than a format issue, [V11N28 Dave Brower] [V11N32 Joseph S. D. Yao] especially considering that there are options to find in 4.2BSD, [V11N24 Jim Cottrell] SunOS 3.2, [V11N36 John Gilmore] and System V Release 3.0 [V11N35 Mike Akre] that take care of this.

3. The default block size in many tar implementations is too large for some tape controllers to read [V11N27 Rob Lake] (the 3B20 has this problem). This is not a problem with the interchange format, however.

There is nothing that the proposed cpio can handle that the tar-based format already in POSIX §10.1 cannot handle; in fact, the former is less capable. If cpio format were augmented to handle missing capabilities, it would be subject to the same objections now aimed at the format given in §10.1: that it was not identical with an existing format.

There is no advantage in replacing the current tar-based format of §10.1 with cpio format. There is also no advantage in adding cpio format, because two standards are not as good as a single standard.

Some have recommended removing §10.1 from POSIX altogether, [V11N12 Doug Gwyn] perhaps with a recommendation for P1003.2 to pick up the idea. [V11N09 Guy Harris] While I believe that that would be preferable to adding cpio format, whether or not tar format remains, I recommend leaving §10.1 as it is, because:

- The inclusion of an archive/interchange file format is in agreement with the purpose of POSIX to promote portability of application programs across interface implementations. Some format will be used. It is to the advantage of the users of the standard for there to be a standard format.

- The de facto standard is tar format. The current §10.1 standardizes that, and provides upward-compatible extensions in areas that were previously lacking.

The Archive/Interchange File Format should be left as it is.

Thank you,

John S. Quarterman
Institutional Representative from USENIX
usenix!jsq

= = =

At its June meeting in Seattle, the P1003 committee decided to put off a decision on formatting until its September meeting. In the interim,

the present cpio format is included in the next draft of the standard, preceded by a note saying that the Working Group must decide which (none, either, or both) will be in the standard, and that a revised proposal is forthcoming.

The options appear to be the obvious:

1. Leave the issue to P1003.2 and remove section 10 from POSIX.
2. Include only ustar format in POSIX.
3. Include only extended cpio format in POSIX.
4. Include both ustar and extended cpio formats as options in POSIX.
5. Require both ustar and extended cpio formats in POSIX.

= = =

The next issue of :login: will continue this discussion. There will be a POSIX implementors workshop in October, see page 3 for the Call for Papers. – PHS

How To Write a UNIX Daemon

Dave Lennert

Hewlett-Packard Company
hplabs!hpda!davel

ABSTRACT

On UNIX systems users can easily write daemon programs that perform repetitive tasks in an unnoticed way. However, because daemon programs typically run outside a login session context and because most programmers are unfamiliar with designing a program to run outside this context, there are many subtle pitfalls that can prevent a daemon from being coded correctly. Further, the incompatibilities between various major UNIX variants compound these pitfalls. This paper discusses these pitfalls and how to avoid them.

Daemon: runs around in the shadows (background) doing devilish deeds.

found in some daemon source code

Introduction

A daemon is a program which performs periodic tasks in such a manner that it is normally unnoticed by users.

Some daemons run constantly, waiting for a significant event. Examples include `init` which respawns login sessions (`gettys`) as they end, `cron` which launches programs at specified times, and `sendmail` which listens on a socket for incoming mail messages.

Other daemons are launched periodically and terminate after completing one execution of their task. Such daemons include the `uucp` file transfer utility, `uucico`, which can be launched as a login shell when a remote machine logs in, `calendar` which is launched nightly by `cron` to examine users' calendars and mail them notification of upcoming events, and various `mail` handling utilities which allow the user's shell to continue while the collected mail message is delivered asynchronously.

Daemon programs are very easy to write in the UNIX environment. They can be written by casual users and launched periodically via the `at` command or, on System V, by a user's personal `crontab` file, or perhaps at each login via `csh`'s `.login` command file. System administrators write daemons whenever they recognize a particular administrative task is becoming routine enough to be handled automatically.

However, daemon programs appear easier to write correctly than they really are. This is because there are many quirks and side effects of UNIX which are automatically taken care of in a login session context but not in a detached, daemon program. The `init`, `getty`, `login`, and shell programs oversee such functions as setting up user ID's, establishing process groups, allocating controlling terminals, and managing job control.

If a daemon process is launched outside a login session (e.g., via `/etc/rc` or a similar function during system startup) then it needs to manage these functions itself explicitly. If a daemon process is launched from within a login session (e.g., as a background command from a login shell) then it needs to undo much of what the login process sequence has done. In order to code a daemon robustly, both concerns must be addressed.

This paper discusses these concerns and the methods for addressing them. Note that all the example coding fragments lack necessary error condition checking or handling; such handling should, of course, be added to any real daemon.

:login:

Programming Rules

The following is a set of programming rules which avoid several subtle pitfalls. A discussion of each pitfall is also given along with the rule.

Make immune to background job control write checks.

On systems which support 4.2BSD style job control, daemons which attempt I/O to their controlling terminal will stop if they were launched from `csh` in the background (with `&`). The real way around this is to disassociate yourself from your controlling terminal (see below). In some cases though, the daemon will want to perform some setup checks and output error messages before it loses its controlling terminal.

There is no way to allow a background process to read from its controlling tty. However, output can be reliably performed if the calling process ignores the `SIGTTOU` signal, as in:

```
#ifdef SIGTTOU  
signal(SIGTTOU, SIG_IGN);  
#endif
```

For safety, it is probably a good idea to ignore the other stop signals as well, as in:

```
#ifdef SIGTTIN  
signal(SIGTTIN, SIG_IGN);  
#endif  
  
#ifdef SIGTSTP  
signal(SIGTSTP, SIG_IGN);  
#endif
```

Ignoring `SIGTTIN` also has the side effect of causing all background attempts to read from the controlling terminal to fail immediately and return the `EIO` error.

Close all open file descriptors, especially `stdin`, `stdout`, `stderr`.

Do not leave stray file descriptors open. More importantly, if any of the file descriptors are terminal devices then they must be closed to allow proper reset of the terminal state during logout (see below). The typical code sequence is:

```
for (fd = 0; fd < _NFILE; fd++)  
    close(fd); /* close all file descriptors */
```

Disassociate from your process group and controlling terminal.

Daemons launched during a login session inherit both the controlling terminal and the process group of that session (or, in the case of job control, of that job within the session).

As long as the daemon is still in the process group associated with a controlling terminal it is subject to terminal-generated signals (such as `SIGINT` or `SIGHUP`). As long as the daemon still has a controlling terminal it is subject to job control terminal I/O restrictions on systems which support job control.

Further, while the daemon remains in the original process group in which it started, it is subject to any signals sent to that process group by another program via `kill(2)`.

One way to prevent the daemon from receiving these “unintended” signals is simply to ignore all signals. However, this means that the signals cannot be used by the daemon for other purposes (such as rudimentary interprocess communication). Also, this approach is insufficient because there are some signals which a process cannot ignore (for example, `SIGKILL` or `SIGSTOP`).

A better approach is for the daemon to disassociate itself from both the controlling terminal and from the process group which it inherited. On 4.2BSD systems, the former can be performed via the `TIOCNOTTY ioctl(2)` and the latter via `setpgrp(2)`. Under AT&T UNIX, `setpgrp(2)` performs both functions.

:login:

However, (under AT&T UNIX) in order for `setpgrp(2)` to have its desired effect, this must be the first time the process has called `setpgrp(2)`; that is, the process must not already be a process group leader. (A process group leader is a process whose process group ID is equal to its process ID.) Since a program has no control over the process which `exec(2)`'d it, it must `fork(2)` to ensure that it is not already a process group leader before calling `setpgrp(2)`. (This is especially important if the daemon is launched from a `csh` which supports job control since `csh` automatically makes its children process group leaders. But this also happens, for example, when an imprudent user launches a daemon from a login shell via the `exec` command.)

In order to prevent locking up a user's terminal when a daemon is started (i.e., without '&'), the daemon usually `fork(2)`'s anyway and runs in the child while the parent immediately `exit(2)`'s without waiting for the child. This causes the shell to believe that the daemon has terminated.

A typical code sequence would be:

```
if (fork() != 0)
    exit(0);      /* parent */

/* child */
#ifndef BSD
    setpgrp(0, getpid());      /* change process group */
    if ((fd = open("/dev/tty", O_RDWR)) >= 0) {
        ioctl(fd, TIOCNOTTY, 0); /* lose controlling terminal */
        close(fd);
    }
#else /* AT&T */
    setpgrp();    /* lose controlling terminal & change process group */
#endif
```

Do not reacquire a controlling terminal.

Once the daemon is a process group leader without a controlling terminal (having called `setpgrp(2)` as described above) it is now potentially capable of reacquiring a controlling terminal. If it does, other processes (for example, logins) will not be able to acquire the terminal correctly as their controlling terminal.

(Interestingly, this problem does not exist under 4.2BSD. Unlike AT&T UNIX, where a terminal can only be acquired as a controlling terminal if it is not already a controlling terminal, 4.2BSD allows a process to join an already allocated controlling terminal and its process group. Basically, the process merges with the already established process group.)

The symptoms of this problem are somewhat subtle. Since `getty` and `login` are not able to acquire a controlling terminal, the special file, `/dev/tty`, cannot be successfully opened. Because of this, the `getpass(3)` routine, used by `login` to obtain the user's password, fails without ever printing the Password: prompt. All login attempts for accounts with passwords silently fail without ever prompting for a password. Login attempts for accounts without passwords succeed (because `getpass(3)` is never called), however the login shell does not have a controlling terminal. Terminal input and output still succeeds (via `stdin`, `stdout`, and `stderr`), but any keyboard signals are not sent to the processes spawned during this login session. Instead the signals are sent to the process which acquired this terminal as its controlling terminal (the daemon) and its descendants.

For this reason the daemon program must ensure that it does not re-acquire a controlling terminal.

On 4.2BSD systems, a new controlling terminal can only be acquired by a process with a process group ID of zero. After calling `setpgrp(2)` to set its process group ID equal to its process ID, the daemon cannot re-acquire a controlling terminal.

:login:

Under AT&T UNIX, a new controlling terminal is acquired whenever a process group leader without a controlling terminal opens a terminal which is not already the controlling terminal for another process group. On such systems the daemon can reacquire a controlling terminal when opening, say, */dev/console*, to perform logging or error reporting. Even if the daemon subsequently closes the terminal it still possesses it as a controlling terminal. There is no way to relinquish it since subsequent *setpgrp(2)* calls are ineffective. (*setpgrp(2)* has no effect if the caller is already a process group leader.) Therefore the acquisition must be prevented.

One simple way to prevent the acquisition of a new controlling terminal is to *fork(2)* yet another time *after* calling *setpgrp(2)*. The daemon actually runs in this second child and the parent (the first child) immediately *exit(2)*'s. However, on AT&T UNIX when the parent (first child) terminates, the SIGHUP signal is sent to the child since the parent is a process group leader. Thus, the parent must ignore SIGHUP before *fork(2)*'ing the second child otherwise the child will be killed. (The ignored setting is inherited by the child.) The final side effect of the terminating (process group leader) parent is to set the process group of the child to zero. The daemon (second child) now has no controlling terminal, it is in a new (zero) process group which is immune to signals from the tty driver, and it cannot acquire a new controlling terminal since it is not a process group leader.

Thus the typical code sequence becomes:

```
if (fork() != 0)
    exit(0);           /* first parent */

/* first child */
setpgrp(); /* lose controlling terminal & change process group */
signal(SIGHUP, SIG_IGN); /* immune from pgrp leader death */
if (fork() != 0)          /* become non-pgrp-leader */
    exit(0);           /* first child */

/* second child */
```

Do not "hold" open tty files.

Even after ensuring that the daemon will not re-acquire a controlling terminal when a terminal device is opened, there is a further concern:

Terminal state settings, such as BAUD rate and signal character definitions, are only reset to the default state when the last process having the terminal open finally closes it. Thus, if the daemon has a terminal open continuously, then the last close never happens and the terminal settings are not reset at logout.

Typical examples of terminal files held open by a daemon are *stdin*, *stdout*, *stderr*, and */dev/console*.

It's probably best to log errors and status messages to a disk file rather than a terminal. However, when terminal logging is desired, the "correct" method is to hold the terminal open only long enough to perform a single logging transaction. Note that this logging transaction still represents a window of time during which a logout would not reset the terminal state.

4.2BSD systems have a further problem which makes this suggestion mandatory. Whenever a new login session is initiated via *getty* or similar routine, the *vhangup(2)* system call is invoked to prevent any existing process from continuing to access the login terminal. This results in read and write permissions being removed from any currently open file descriptor which references the login terminal; this affects all processes regardless of user ID. Therefore, daemons which access a terminal that is also used for regular login sessions, must reopen it whenever access is desired. If a file descriptor for such a terminal is continuously held open, it is very likely that *vhangup(2)* will quickly destroy its usefulness.

To determine if an unknown file descriptor is a terminal device use *isatty(3)*.

:login:

Change current directory to '/'.

Each process has a current working directory. The kernel holds this directory file open during the life of the process. If a process has a current directory on a mounted file system, the file system is "in use" and cannot be dismounted by the administrator without finding and killing this process. (The hard part is finding the process!) Unless a process explicitly alters this via `chdir(2)`, it inherits the current directory of its parent. When launched from an interactive shell, the current directory will be whatever the user has most recently selected via the `cd` command.

Because of this, daemons should adopt a current directory which is not located on a mounted file system (assuming that the daemon's purpose allows this). The root file system, '/', is the most reliable choice. The simple call is:

```
chdir("/");
```

Reset the file mode creation mask.

A file mode creation mask, or `umask`, is associated with each process. It specifies how file permissions are to be restricted for each file created by the process. Like the current directory, it is inherited from the parent process and remains in effect until altered via `umask(2)`. When launched from an interactive shell, the `umask` will be whatever the user has most recently selected via the `umask(1)` command.

A daemon should reset its `umask` to an appropriate value. The typical call would be:

```
umask(0);
```

Other attributes to worry about.

The environment attributes discussed above are the primary ones to worry about, but the list is not exhaustive. Any attribute inherited across an `exec(2)` system call is of concern. Some other calls to be cautious of are the nice priority value (see `nice(2)`), the time left until an alarm clock signal (see `alarm(2)`), and, on 4.2BSD systems, the signal mask and set of pending signals (see `sigvec(2)`). However, these are less likely to be *accidentally* set "wrong."

Interactions With `init`

The system initialization process, `init`, is responsible for directly or indirectly starting all processes on the system (with the exception of kernel processes such as the swapper or pageout process). On many versions of UNIX, `init` keeps track of all processes which it directly spawned and it can optionally respawn them if they die or it can kill them when changing to a new system *run state* (or *level*). Under AT&T UNIX, the `/etc/inittab` file specifies the programs `init` should spawn in which run levels and whether or not they should be automatically respawned when they die. (Note that this file differs in both format and capabilities between System III and System V.)

Historically, system daemon programs are launched by the `/etc/rc` shell script which `init` launches when moving the system from the single user run state to multi-user mode.

Some system administrators now prefer to launch daemons directly from `init` by placing the appropriate commands in `/etc/inittab`. They rely on `init` respawning the daemon should it inadvertently die and on `init` killing the daemon during system state changes.

Note that the respawning and terminating capabilities of `init` depend on the spawned program not terminating prematurely. The above programming rules, however, suggest that daemons should immediately `fork(2)` and have the original process `exit(2)`. If launched from `/etc/inittab`, this procedure would cause `init` to believe that the daemon was no longer running and hence it would not terminate the daemon during state changes and would instead immediately relaunch the daemon (if automatic respawn were requested). This procedure thus defeats both the respawning and terminating capabilities provided by `/etc/inittab`.

:login:

What can be done to correct this? The only solution is to prevent the daemon from following the above procedure if it is launched from */etc/inittab*.

One tempting approach is for the daemon to retrieve the process ID of its parent immediately using `getppid(2)` and, if it is `init`'s process ID (1), skip the problematic code. However this is not perfectly reliable since any process whose original parent has terminated assumes `init` as its new parent. If a daemon is launched interactively from a user's shell, the shell might subsequently terminate before the daemon has executed the `getppid(2)` call. In short, there is a race condition. However, for practical purposes, this is a quick and easy way to solve the problem.

Another approach is to pass a command line flag to the daemon indicating whether the daemon is being launched from */etc/inittab* or not. But this requires the user to set the flag correctly during both automatic and interactive invocations. A common error would be for a user to examine the launching command in */etc/inittab* and then use it verbatim interactively.

Regardless of what approach is used, all the above mentioned pitfalls must still be recognized and avoided.

In the final analysis it seems that launching daemons from */etc/inittab*, as opposed to */etc/rc*, is unnecessary for the following reasons: (1) Relying on `init` to respawn a daemon is really masking a bug in the daemon; the daemon should never terminate by itself.[†] (2) Changing run states is an unusual occurrence on most systems; usually a system will move to multi-user mode and stay there.

Conclusions

Without following the above rules, strange symptoms which are hard to track down often result. Many times the errant daemon program is the last thing suspected (e.g., when terminal settings are not reset after logout). Other times it is the daemon that silently and mysteriously dies (e.g., when it attempts background I/O on a job control system). Frequently these symptoms only begin occurring well after the "debug period" for the daemon.

Example

The example below collects the above coding fragments into a single routine which a daemon calls to detach itself from the context of a login session.

```
/* Detach a daemon process from login session context.
 *
 * (This is a skeleton; add error condition checking and handling.)
 */
#include <signal.h>
#include <stdio.h>
#ifndef BSD
#include <sys/file.h>
#include <sys/ioctl.h>
#endif
sessdetach()
{
    int fd;           /* file descriptor */

    /* If launched by init (process 1), there's no need to detach.
     *
     * Note: this test is unreliable due to an unavoidable race
```

[†] One interesting counterexample is that some systems (e.g., ACSnet) allow system administrators to reset things by killing the appropriate daemons. It's nice to have the daemon start correctly (i.e., right arguments) by itself through the auspices of */etc/inittab*. However, it's arguably better to have the daemon catch the termination signal and perform the reset without actually terminating; this may even be essential in the case of orderly shutdown of operations such as line printer spooling.

:login:

```
* condition if the process is orphaned.  
*/  
if (getppid() == 1)  
    goto out;  
/* Ignore terminal stop signals */  
#ifdef SIGTTOU  
    signal(SIGTTOU, SIG_IGN);  
#endif  
#ifdef SIGTTIN  
    signal(SIGTTIN, SIG_IGN);  
#endif  
#ifdef SIGTSTP  
    signal(SIGTSTP, SIG_IGN);  
#endif  
/* Allow parent shell to continue.  
 * Ensure the process is not a process group leader.  
 */  
if (fork() != 0)  
    exit(0); /* parent */  
/* child */  
/* Disassociate from controlling terminal and process group.  
 *  
 * Ensure the process can't reacquire a new controlling terminal.  
 * This is done differently on BSD vs. AT&T:  
 *  
 *      BSD won't assign a new controlling terminal  
 *      because process group is non-zero.  
 *  
 *      AT&T won't assign a new controlling terminal  
 *      because process is not a process group leader.  
 *      (Must not do a subsequent setpgrp())!  
 */  
#ifdef BSD  
    setpgrp(0, getpid()); /* change process group */  
    if ((fd = open("/dev/tty", O_RDWR)) >= 0) {  
        ioctl(fd, TIOCNOTTY, 0); /* lose controlling terminal */  
        close(fd);  
    }  
#else /* AT&T */  
    setpgrp(); /* lose controlling terminal & change process group */  
    signal(SIGHUP, SIG_IGN); /* immune from pgrp leader death */  
    if (fork() != 0) /* become non-pgrp-leader */  
        exit(0); /* first child */  
    /* second child */  
#endif  
  
out:  
    for (fd = 0; fd < _NFILE; fd++)  
        close(fd); /* close all file descriptors */  
    chdir("/"); /* move current directory off mounted filesystem */  
    umask(0); /* clear any inherited file mode creation mask */  
    return;  
}
```

Book Reviews

The Design of the UNIX Operating System

by Maurice J. Bach

(Englewood Cliffs, NJ: Prentice-Hall, 1987) 471 pages, \$31.95

Reviewed by Marc D. Donner

IBM Thomas J. Watson Research Center
ucbvax!ibm.com!donner

There are four potential audiences for this book. The first potential audience is a class of students in an operating systems course. The second potential audience is UNIX system hackers interesting in deepening their understanding of the internals of UNIX. The third potential audience is application programmers in the UNIX world who think a better understanding of how the system works will help them do a better job (it will). The fourth potential audience is everyone with a prurient interest in finding out what goes on under the covers, with no other goal than improving his understanding.

The version of UNIX that Bach focuses on in this book is System V Release 2, but he also discusses some of the major Berkeley improvements, though not the file system.

As an operating systems text, I give this book a B. It earns a solid A for the second and third audiences and it earns a stellar A+ for the fourth audience. It suffers from a scattering of minor errors, both technical and typographical, but with one exception they are insignificant. The rest of this review consists of an evaluation of the book's merits for each of the potential audiences, a summary of the contents, and a brief precis of the significant errors.

As an Operating Systems Textbook

In many ways this book is attractive as a text for an operating systems course. It is remarkably well written, the implementation of UNIX internals is clearly explained, and the exercises are generally well designed. Its weakness as a text is that it is very UNIX-centric. In many places the exposition focuses on how it is done in UNIX-as-it-is rather than on what the alternatives are in the general operating system design context. The book is not terribly strong

on historical perspective, nor on citation to the relevant non-UNIX literature. This is a particularly dangerous blind spot for a text, since it can give impressionable students a "there is one way and Ken and Dennis are the prophets" view of operating system technology. This book might be an excellent adjunct to an advanced operating system course, with general principles taken from some other text or, better yet, from a collection of the important papers in the field.

As a Text for UNIX System Programmers

This book is the shortest path to a solid understanding of the important issues in the UNIX kernel that I can imagine. Its completeness is amazing, something that is difficult to appreciate without spending several hours reading and referring to the book. The explanations of the various kernel functions, which Bach reduces to pseudo-code and calls "algorithms," are clear and complete. There is good attention to detail, including careful analysis of race conditions that motivate various details of the implementation.

The book is an excellent item to have by your side when digging into some ugly piece of kernel code. It has helped in the deciphering of more than one piece of typically impenetrable and inscrutable source.

As a Text for UNIX Application Programmers

The understanding of the underlying design and implementation of the UNIX kernel will help any application programmer make better design decisions. This book doesn't give any instruction in the writing of application programs, but it does demonstrate the working of many kernel calls, with excellent exposition of their

properties. Eccentric behavior is exposed and explained, something that the manual pages are notorious for not doing.

Without descending unbearably into horrible detail, the book manages to illustrate the functioning of most of the interesting facilities. The discussion of signals in section 7.2 is excellent. It cleared up a number of problems in my own understanding of what was going on with signals.

As an Expose of the Internals for UNIX Lovers

This is where the book shines best. It is so comprehensive and generally so clear that it is a delight to read. The reader really should examine all the algorithms carefully, and it helps to work the examples to get the most from the book. There is a wealth of sample programs that highlight various odd or interesting behaviors. Most are worth typing in and executing, though some of the most interesting require superuser privileges.

The section that explains how demand paging works gives a good explanation of how to do it on a machine without a reference bit, tactfully refraining from any comment on machines with such a lack.

Chapter Summary

Chapter 1 is an introduction, with some history and philosophy. Chapter 2 gives an overview of the structure of the kernel, explaining the major sections and the division of responsibilities among the file system, the process control system, the I/O system, and the other components. Chapter 3 examines the buffer cache in detail. This chapter contains the only serious technical error in the book. The algorithm `getblk` has an error in it as presented in figure 3.4. The problem is that the original C code has a slimy interlocked loop, but the translation presented in the book has lost the function by simplifying it incorrectly. The text on page 48 describes what should happen correctly and a minor change to the algorithm will make it work, but it took two of us quite a while, plus a visit to our source code, to verify the error.

Chapter 4 contains more than you ever thought possible about the representation of files. The only topic that I can think of that it neglects is recent improvements that result in better clustering of blocks in the file system. Chapter 5

explores the file system calls, explaining the functioning of each one and relating it to the underlying data structures described in the previous chapter.

Chapter 6 begins the second half of the book with a detailed description of the structure and composition of processes. Process execution state, context, and memory are covered in detail. This is probably the trickiest chapter of the book. Chapter 7 covers process control, with `fork`, `exec`, `pipe`, `dup`, and other system calls being covered in detail. Signals are beautifully covered in this chapter as well. The exercises for this chapter are particularly numerous and challenging. Chapter 8 talks about process scheduling, with the time-related system calls thrown in for good measure.

Chapter 9 discusses memory management, both swapping and paging, in reasonable detail. There is a misleading statement in the introduction of the chapter that might cause a naive reader to think that 4.0 BSD was the first implementation ever of a demand paging policy, when it was really just the first UNIX system with demand paging. Chapter 10 talks about the I/O subsystem, with an explanation of the architecture of device drivers and some discussion of various device types. Chapter 11 is entitled “Interprocess Communication” and discusses both System V IPC and BSD Sockets. Chapter 12 discusses some of the issues involved in building a multiprocessor UNIX, while Chapter 13 talks about distributed versions.

There is an appendix containing a summary of all the system calls, with brief descriptions of what each does. The bibliography is fairly extensive, though it is focussed primarily on UNIX. The index is good but not great.

Important Errors in Brief

Exercise 2 on page 37 has the names of `bp` and `bp1` swapped from the names used in figure 2.7. Figure 3.4 on page 44 has a serious error, already discussed above in the section on Chapter 3. Figure 5.20 and the text describing it do not agree. The hyphen breaking the word “Superusers” on page 121 extends into the margin of the page. The description on page 205 of figure 7.9 refers to an address as hexadecimal ee, but it should say f9. In figure 7.34 the first two occurrences of the variable `srcfile` should be replaced by `argv[1]` and `argv[2]` respectively. The swapper algorithm described

on page 281 is not quite right for the revised algorithm.

There are several stylistic and presentation problems as well. The table of contents is presented double spaced and in ALL CAPITALS, thus making it difficult to browse. In addition, it seems to me that there is really a structure over the chapter structure that should be exhibited by breaking the book into parts, with chapters 1 and 2 in the first part, chapters 3 through 5 in the second part, chapters 6 through 8 in the third part, chapters 9 and 10 in the fourth part, and chapters 11 through 13 in the last part. The exhibition of this structure in the table of contents would help illustrate the system structure that it mirrors. Another minor complaint: the publisher yielded to the modern

temptation to slip junk in at the end of the book, thus inserting four blank pages and a page of advertizing in between the index and the end paper. This is a recent disease among publishers that should be strongly discouraged, since it reduces the utility of the index of a book. There should be absolutely nothing between the index and the end paper. Nothing.

Summary

This is a truly outstanding book and it belongs in the library of every serious UNIX programmer. System hackers will find it invaluable, while application programmers will find it at least very interesting and informative. I think that it does not make an outstanding operating system textbook, but it would be an excellent adjunct to an operating system course.

= = =

A C Reference Manual (Second Edition)

by Samuel P. Harbison and Guy L. Steele, Jr.

(Englewood Cliffs, NJ: Prentice-Hall, Inc., 1987) \$31.00 hardcover, \$24.95 paper

Reviewed by Josiah C. Hoskins

joho@mcc.com

I think most everyone would agree that the definitive C reference book for the past decade has been *The C Programming Language* by Kernighan and Ritchie (1978). However, in 1984 the first edition of *A C Reference Manual* by Harbison and Steele appeared which provides an even more detailed and current description of the C language. I am not proposing that Harbison and Steele's reference manual is a replacement for K&R's *The C Programming Language*. I am stating that Harbison and Steele's book provides an excellent companion to K&R which reflects the fact that C is a dynamic and changing language and reference manuals must change as the C language matures. I attribute the popularity of this reference manual to four assets: 1) it has provided a current reference for the C programming language (filling the gap since 1978), 2) it reflects common interpretations of C in different machine environments, 3) it reflects the current ANSI

standardization effort, 4) it provides one of the most complete, accessible, and well written references to the C programming language.

The second edition has varied little from its original intent to provide a detailed description of the current C language. Several changes have taken place that may encourage a C programmer to fork over the approximately \$31.00 (hardcover, \$24.95 paper) for the 2nd edition. However, the first change one notices is the smaller point size of the type face. This is not so nice as I program late and the smaller the print the less I see. Cheers, in section 1.3 Syntax Notation we find that that ^&%\$ Backus-Naur Form (BNF) notation has been dropped! The first obvious substantive change is that there are many more references to suggestions of the Draft Proposed ANSI C Standard. The References now contain pointers to Draft Proposed ANSI C - included in chapter eleven.

Unlike some second editions it's obvious that the whole book was read for content and modified as needed. Of course, the homey character graphics figures remain and there still exists a wealth of example code. A welcome improvement is the expanded treatment of the character type (important in portability). As in the first edition chapter five on data types is worth the price of the book. Chapter six contains a new section entitled Representational Issues. It covers addressing structure and byte ordering, I mean where can you find that kind of information. The same precedence chart (note p.141 in both editions) is there again; give me p.49 in K&R. Some chapters like eight and nine hardly changed at all. Small changes were found in the stack example of chapter ten to better demonstrate program structure.

The rest of the second edition is quite different in both organization and somewhat different in content. Chapter eleven is a

totally new chapter which summarizes Draft Proposed ANSI C by specifying how it differs from the C language as presented in H&S. The run-time library chapter has been completely reorganized and contains additional coverage of topics such as signals, time of day functions, and operations on arbitrary blocks of memory. Appendix B contains a revised syntax of the C language which includes both the traditional and Draft Proposed ANSI C. It is a nice presentation with both the Lexical rules and the Draft Proposed ANSI C clearly marked in the text.

As you should have guessed, I like the H&S *C Reference Manual* and have made good use of the first edition. If you already have the first edition, the second edition will not change the way you program significantly, however, it will provide you an up-to-date status of C. If you don't have a copy of *A C Reference Manual* I recommend getting one and you will find yourself writing more clear, correct, and more importantly portable C programs.

= = =

UUNET Progress Report

At the Winter '87 USENIX Conference in Washington, DC, the USENIX Association announced the funding of the UUNET project on an experimental basis. UUNET became operational in mid-May. As of July 1 UUNET has over 50 subscribers. This article introduces UUNET to the USENIX membership and provides a progress report.

UUNET is a non-profit communications service that provides access to USENET news, UUCP mail, and ARPANET mail. UUNET is the new experimental project of the USENIX Association with the unprecedented cooperation of DARPA.

For this experiment, DARPA has authorized the use of the Center for Seismic Studies personnel, resources, and communications facilities. This allows UUNET to house its host computer at a well-staffed and maintained computer center and to provide the high quality services necessary for this project. In addition, DARPA has authorized

use of the ARPANET gateway at the Center on an experimental basis to test the feasibility of mail forwarding between ARPANET and non-ARPANET sites.

This is the first time a joint project like this has been initiated and the experiment will be carefully conducted to assure that all ARPANET and Center policies are followed. The technical results of the experiment will be presented to DARPA for their consideration of the long-term possibilities of continued interaction and to USENIX for their funding consideration.

The USENIX Association will provide financial and administrative support during the course of the experiment. In addition to news and mail, UUNET will provide access to various source archives, many standards, (including the Internet RFC's and comp.std.unix archives), and NIC service for the USENIX community.

There are no restrictions on what you may send nor on redistributing what you obtain from or through UUNET. UUNET is effectively a common carrier. Charges are calculated to recover costs.

UUNET exists as a communications relay. It is not used for any other project and is maintained 24 hours per day. The number of intermediate hops to members for news and mail is greatly reduced, thereby increasing the reliability. In the first month that UUNET was operational, there was no unscheduled downtime.

Subscribers can be directly connected to a backbone site and not have to depend on others to redistribute newsgroups. Subscribers may have a full newsfeed, a partial newsfeed, or none at all. (A full newsfeed costs about \$175 per month in connect time.) UUNET always carries all newsgroups. This includes any new news categories that may appear other than those in the standard set.

UUNET makes available for UUCP access an extensive archive of publicly available UNIX software. This includes the latest GNU software, the latest Kermit distributions (for many CPU types, not just UNIX), all the Internet RFCs, the latest UUCP map information (updated daily from the master copy), access to the Simtel-20 archives, and the netlibd archives at Argonne (EISPACK, LINPACK, etc.).

Operationally, UUNET consists of a 10 processor Sequent Balance 21000 located at the Center for Seismic Studies at Arlington, VA. The system is connected to Tymnet via a high-speed leased line. It can handle 25 simultaneous uucico transfers and will be upgraded to match demand. It is administered by the same people who are currently administering *seismo*. Operations personnel are on site 24 hours/day Monday-Friday and someone is always on call on weekends. Currently, the UUNET machine is tightly coupled to *seismo*. This means that having a connection to UUNET is effectively having a connection to *seismo*, i.e., a well-connected news and mail relay.

To access the UUNET system from within the United States, subscribers dial a local phone number (from thousands of US cities) and connect to Tymnet. Subscribers are then connected to UUNET via the Tymnet X.25 public data network. International sites may access UUNET via direct host-to-host X.25 connection. No special hardware or software is required (other than the standard UUCP protocols). The connection to Tymnet is made with an ordinary modem (V.22bis / Bell 212 / Bell 103).

The cost of using UUNET is \$3 per hour of connect time during off-peak times (\$5 per hour from Hawaii). Off-peak times are 6:00 p.m. to 7:00 a.m. Monday-Friday and all day Saturday and Sunday. (The subscriber's time zone is used to determine peak or off-peak time, not necessarily the time zone in which the UUNET system is located.) UUNET accepts calls 24 hours a day. Access is available during peak rate time at substantially higher rates (\$20-\$32 per hour depending on location). There is a membership charge of \$30 per month to cover administrative costs.

As previously mentioned, USENIX has funded UUNET for an experimental period. UUNET expects to offer these services at these prices by generating a large volume of traffic. If a large enough volume of traffic is seen by the end of the experiment, USENIX will spin off the UUNET experiment into an independent, non-profit organization that will continue the service with the same basis. If a large enough interest is not shown to allow UUNET to recover its operating costs, USENIX will regrettably have to discontinue funding.

For further information on UUNET, please contact:

Peter Salus
UUNET / USENIX
P.O. Box 2299
Berkeley, CA 94710
+1 415 528-8649
(seismo, uunet, ucbvax, cbosgd, ames,
amdahl)!usenix!peter



UK UNIX® systems User Group



FaceSaver 6/87

Sunil K Das

Chair

The City University,
Computer Science Dept.,
Northampton Square,
London EC1V 0HB.
Tel: 01-253 4399 ext: 3725

Network Address:
sunil@cs.ucl.ac.uk
ukuug@ukc.ac.uk

PRELIMINARY ANNOUNCEMENT and CALL FOR PAPERS

EUUG SPRING '88 CONFERENCE UNIX AROUND THE WORLD

London, 11-15 April, 1988

Preliminary Announcement

The UKUUG will host the Spring '88 European UNIX systems User Group Technical Conference in London. Technical tutorials will be held on Monday 11th and Tuesday 12th April followed by the three day conference, ending on Friday 15th April.

A pre-conference registration pack containing detailed information will be issued in early December 1987.

Theme

The conference will draw technical papers from every continent where UNIX licensees exist. The highest quality papers from as many different countries as possible will be refereed for the EUUG Conference Proceedings. Teleconferencing by Satellite will stress the truly global quality of this gathering. The UKUUG's intention, on behalf of EUUG, is to provide an international forum for presentation and exchange of current work on a wide variety of topics related to the UNIX system and C language.

Call for Papers

The EUUG invite abstracts from those wishing to present their work. All submitted papers will be refereed. They will be judged with respect to their quality, originality and relevance.

Suggested topics include, but are not limited to:

- Programing Environments and Tools.
- Communications.
- Recent work in Standards and Portability.
- Real-time UNIX.

© Unix is a registered trademark of AT&T in the USA and other countries

Submissions from Students are particularly encouraged under the EUUG Student Encouragement Scheme, details of which are available from the EUUG Secretariat.

Important Dates

Abstract Deadline	30th October 1987
Acceptance Notification Posted	20th November 1987
Final Paper received	15th January 1988

Method of submission

Abstracts must be submitted by post to the EUUG Secretariat at the address below. All submissions will be acknowledged by return of post.

Tutorial Solicitation

Tutorials are an important part of the EUUG's biannual events providing detailed coverage of a number of topics. Past tutorials have been taught by leading experts.

Those interested in offering a tutorial should contact the EUUG Secretariat as soon as possible.

Secretariat

The EUUG Secretariat may be contacted at:

EUUG
Owles Hall
BUNTINGFORD
Herts SG9 9PL
UK

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

Additional Information

The Programme Chair, Sunil Das, is willing to provide advice to potential speakers. He may be contacted at the address shown on the front page.

Venue

The conference will be held in the prestigious Queen Elizabeth II Conference Centre in London. This venue embraces over 13 centuries of history represented by Westminster Abbey, the Houses of Parliament and the Palace of Westminster all within the sound of Big Ben. Conveniently located close to London's theatres, St James' park and the world famous cultural complex on the South Bank.



Sunil Das
44-1-253-4399
City University - London
Computer Science Dept.
Northampton Square
London, United Kingdom EC1V 0HB

**EUROPEAN
UNIX SYSTEMS USER GROUP
NEWSLETTER**

*Volume 7
Number 1*

The EUUG Newsletter: A Wind of Change	1
UNIX Conference Reports	
Atlanta Usenix, June 1986	3
The Manchester Competition	19
Uniforum, January 1987.....	25
Notes on the Birth of the UNIX Cult.....	29
The X/OPEN Show in Luxembourg	37
The CV Macros.....	39
News from the National Groups	41
AFUU Diary.....	43
News from the Danish Group.....	45
News from the Netherlands	47
Letter from Germany	49
Forthcoming UKUUG Events.....	51
GKS in C++.....	53
An NRS Processor in C and the Future.....	65

Trip Report: Atlanta USENIX June 1986

Nick Stoughton
n.stoughton@inset.co.uk

The Instruction Set

An amusing but content free diary of events during the 1986 Atlanta USENIX meeting. It is suggested that the reader refers to the conference proceedings for details of the papers actually given!

1. Friday 6th June, 12.00

Summoned to Nigel Martin's office; "Boyd is ill, he will be off sick for at least two weeks. Therefore he will be unable to go to Atlanta on Monday. Would you like to go?"

What a stupid question! Enter panic mode to get things tidied up, flights changed, hotel reservations changed, etc etc. Ticket delivered.

2. Monday 9th June, 11.00

Notice that itinerary has the return flight taking -5 hours, after adjusting for local time differences! Not even Concorde goes that fast! They got the time of landing out by 12 hours. Oh well. Taxi arrived for the airport. Last minute rush to get tapes made for customers. Wave goodbye. Flight is from Gatwick at 13.40. Call in at home (Redhill, just 4 miles from Gatwick) to pick up suitcase. Leave a message for the wife so she doesn't sit at the airport for twelve hours on Saturday.

The plane is crowded. Sitting right at the front of the economy class area, with the bulkhead right in front of me. Too close to the screen to watch the movie. They charge for it anyway, and it will probably be grotty.

World Airways food seems even worse than the average airline rubbish.

3. Baltimore Washington International

Coming in to land, and the condensation forms on the overhead lockers, and pours on the people sitting in the front rows. That's me. Raining inside the aircraft. Brilliant.

300 people pour off the aircraft and wait at the immigration desk. Two IO's, giving all non-nationals a really hard time, especially if they are non-caucasian. Three quarters of an hour later, my turn. Through in about ten seconds. What happened?? Now I have to run for the connection to Atlanta. Collect baggage, more questions from customs. Check baggage in. Get on plane. Relax again. Flight delayed anyway. Local time is 8.00 pm, body trying to tell me that it is 1.00 am.

4. Atlanta, June 9th

As you step off the plane onto the jetway, it feels like they have a warm air heater switched on. Cool again inside the terminal. Collect baggage, step outside.

Brain refuses to accept that it is both dark and over 90 degrees. Over 90% humidity too. Hot and sticky. Horrid. Catch airport shuttle bus at 9.30 (2.30 am). Get to hotel at 10.00 pm.

"I have a room reserved, the name is Stoughton, Nicholas Stoughton."

"Mr Anthony Stoughton?", pronounced "anthhhh owny".

"No, Nicholas. From London. I work for a company called the Instruction Set. My reservation was originally made in the name of Boyd Roberts."

"Let me check that. How do you spell Boyd?"

"B-O-Y-D, surname Roberts."

"Just a moment..."

Ten minutes go by. I am falling asleep on the counter.

"I have no record of a Mr Robert Boyd. Just a Mr Anthhhowny Stoughton, from a company called the Instruction Set in London England."

"That's me...you got the name wrong, but there aren't too many Stoughtons working for the Instruction Set right now." Notice I am getting into the americanisms..."right now". Yuck.

"It says here on the reservation form that you will be arriving late."

They're clever these Americans, they can read. I guess you could say 10.00pm is fairly late to check in.

"We have a small problem Anthhhhowny."

"The name's Nick, but what's your problem?" Not my problem, note.

"Well, when you hadn't arrived by 6.00pm, we let the room to someone else. There is this Southern Baptist Convention on right now, and the city is very busy. We do have another room that you could have though."

My body assures me that it is 3.30 in the morning. Travelling always makes me tired anyway. I really fancy just going to sleep. So I ask

"What's the catch?"

"It really is a very nice room, Sir, very spacious. One of our parlour (parlor??) rooms. But it doesn't have a bed."

Oh no.....

"Of course it doesn't cost as much."

Since I am not paying the bill at the end of it, this is a small consolation. And with the town full of the Suuuuuthern Baaaaaptists, I'm not going to find it easy to get a room elsewhere at this time of night.

"I am intending to be here until Friday. I take it that you will be able to get me a proper room tomorrow?"

"Oh yes, Sir, of course."

"I'll take it, but only under extreme protest, and because if I don't find somewhere to sleep pretty soon, I would sleep in the corridor if necessary. Where is this room without a bed?"

"Well, it does have a folding bed thing, and it is on the 67th floor."

I discover that this is the tallest hotel in the world. And I'm practically at the top. The room actually is huge, and the folding bed is not too uncomfortable. Sleep at last.

5. Tuesday, 9.00

This hotel is full of Southern Baptists. They try to convert you in the elevator (getting used to the jargon...those things that whisk you up to the 67th floor are not called lifts here).

There are about 50 SB's queuing for breakfast. Give this up and go to the Hilton.

Better do the registration bit before eating, just in case there's problems.

Go to counter marked "Pre Registration", because I was pre-registered. Or at least I thought I was.

"I'm sorry, we have no record of a Mr Stoughton."

Oh boy, here we go again.

"Could you try Boyd Roberts, The Instruction Set."

"I'm sorry, nothing for him either. You'll have to go to the On Site registration desk."

Well, not much I can do besides go to the on site registration. Another hundred bucks on the expense claim.

"What tutorials do you want to go on."

"System V internals, please."

"Can I see your source licence?"

I'd left so fast on Monday that I'd clean forgotten to bring a copy. But one should have been sent with the pre-registration, so I wasn't bothered till now.

"I am sorry, I have forgotten to bring it. Anyway, you were sent one a copy with my pre-registration that seems to have got lost. Couldn't you look it up?"

"Our lists are very out of date, I'm afraid. But I'll take a look." He looks under "I" for "Instruction Set". No record.

"But I have got a source license. Would you like me to quote some lines of source from memory or what?"

"I'll take one more look. The Instruction Set, I suppose it could be under "T"...ahh...yes...The Instruction Set, Technical Contact Andy Rutter."

"That's the one. Now can I go to the tutorial?"

"I guess so."

Great. We are finally getting somewhere. Now I can go and eat some breakfast.

6. Tuesday, Tutorial Session

Well, avoided the eggs because I couldn't face the wide variety of choices of cooking method, each with strange names. Now for some real work at last.

Tutorial #T3. UNIX System V Internals, with Maury Bach and Steve Buroff. An interesting mixture of some really simple stuff (how system calls work), and some extremely useful explanation of some of the SVr3 features, the File System Switch,

Remote File Sharing, Streams and the Transport Level Interface. About 75% extremely interesting 10% interesting and 15% boring.

Steve Buroff defined a "property" of a system to be a combination of "a feature and a restriction". A good definition! One property of shared libraries is that they are statically linked. This means that every shared library must have a fixed address range. AT&T will manage the allocation of address ranges for the 3B machines. What will happen for the other machines? Will AT&T be capable of managing this sensibly? (Shame on you Nick for doubting AT&T's management capabilities.)

RFS sounds good till you look underneath. Despite the File System Switch's ability to allow non-standard file system types to exist, RFS seems to have ignored this in slavish adherence to the SVID file standards. This is fine, but virtually prohibits sharing of files with non-UNIX file systems.

They did funny things to *copyout* too.

7. Tuesday Evening

It is raining. Not just drizzle, or light rain. This is a monsoon. Do they have monsoons here? You bet they do. About 6 inches of rain an hour. Sit in the Hilton waiting for a break. Race back to Westin Peachtree so that I can book Nigel in before the 6.00pm deadline. Find out that he has already made it. Start the hunt for him. Not in bar, but had a quick one whilst looking. Back to the Hilton. No sign. Quick check in the bar over a cocktail. Definitely not there. Go for a meal instead. These Americans definitely like to eat lots of food, and extremely tasty too; pity about the beer.

Back to the hotel, move to new room, two floors down. It still takes 3 minutes for the elevator to cover the 65 floors, and the Southern Baptists can get an awful lot of praise into 3 minutes.

8. Wednesday Morning

Finally find Nigel by means of calling his room. Try for breakfast in the hotel. Taxi to the Hilton. Well, maybe it is only two blocks, but it is awful hot out here.

First day of the conference proper. Nigel has had this idea, developed over his previous evening's entertainment with the other EUUG people. "Run the *errno* contest that went down so well in Florence here." The contest is announced during the opening session. Murmurs of approval from the masses.

The keynote address from Jon Bentley is entitled "Pictures of Programs". Not surprisingly, quite a lot of it is about graphics.

Coffee time; find a box for the *errno* contest. Tilbrook tries his hand at sign writing. Not too bad. Now sit back and wait for the entries. Will anyone beat ENOTABACCO, read on an empty pipe?

Mike Hawley presents his MIDI paper, this time round it is a history of Western Music since the Middle Ages, complete with numerous taped examples of everything from Gregorian Chant to some contemporary stuff (can't remember who by though!) Fascinating, but impossible to describe on paper.

Peter Langston follows hot on Mike's heels. Aside from writing games, Peter's main *raison d'être* is music. Music, I believe, of any sort whatsoever. Computer generated music most of all. Working at Bellcore he has of course access to all sorts of nice bits of hardware, and telephone equipment. So via a complex arrangement of Dec-

Talk speech synthesisers and music making synthesisers, anyone with US standard MF dial phone can dial in on (201) 644-2332, and listen to Eddie and Eedie taking you through some generated-on-the-spot music in their comical voices. These machines are truly tremendous. After the initial period, you really don't realise that this is computer generated. The voices have all the right intonation and inflexion, the music is spot on. Simple algorithms imitating a facile, unimaginative and slightly lazy guitarist playing random riffs to fill in bits ensure that the music remains interesting. So interesting that no-one would have minded if he had talked, and played us recordings of Eddie and Eedie, all day.

9. Wednesday Afternoon

Quick lunch from the rather good serve-yourself restaurant in the Hilton foyer, talking to Melvin from BT. These Brits get everywhere.

Back upstairs, the first of the *errno* entries have been submitted.

ENOGOOD	Invalid system error
ECHERNOBYL	Connection melted
EREALSOONNOW	Feature not yet supported
ECREAT	Missing vowel

About 20 so far.

Networks 1 in the Grand Ballroom, Secure Networking in the Sun Environment; A Framework for Networking in System V; and OSI and TCP/IP Protocols on a UNIX System V. Sounds good.

Secure Networking turns out to be all about public key cryptography and the DES encryption standard. Not as interesting as I'd hoped. Maybe I'll read the paper before going to the presentation next time.

The Framework for Networking in a System V presentation basically described the SVR3 TLI implementation. Good stuff, but there isn't SVR3 to hand yet.

The French described what they had done to some weird piece of equipment they call the SM-90 to put OSI and Internet protocols into their System V. A large part of this exercise was to put the 4.2BSD Socket stuff and most of the networking bits into their kernel. Read the paper.

Tea break. Another 30 entries in the *errno* box. Boy, this is beginning to take off. We already have about the same number of entries as we got for the whole of the Florence contest.

ENIH	There's a better way to do that
EXPORT	Feature restricted outside the US
ENOWARP	Out of Dilithium Crystals
EACDC	Wrong type of socket
ETHIOPIA	Out of resources
EGODS	Too many Baptists
EMAIL	Junk mail detected
EAT	You've been hacking too long, time to ...
E	2.71828...
EFLAT	String out of range

EBADDOG Pointer on carpet

The next session looks a bit heavy. Operating Systems 1 or Tools. Tools contains a talk on Pathalias. Definitely to be missed!

Give up after the first presentation. After all, I can read the proceedings later. Let's go and talk to all those interesting people about. Awful lot of Europeans about making the most of the tea.

Already another 20 *errno* entries.

Nigel has decided to move hotel to the Hilton. It is cheaper and easier. I think I will follow. Check out of the Westin Peachtree. Taxi to the Hilton. Cabbie doesn't like having to take me just 2 blocks. Stiches me up with the fare, so I don't tip him. Looks violent. Run for cover inside the hotel.

Only on the 24th floor here. Peanuts. Good view out over the hotel pool though, which is outside on the roof of the 10th floor, and directly below my window.

10. Thursday AM

Another 50 entries in the *errno* contest. This is going to be fun.

EFIXED	No children
ECONSTIPATED	Operation would block
EAT	The computer is going down
EMARCOS	No longer supported
EVITA	Don't cry for me Argentina
EBUNNY	Multi-hop access attempt
ELUSIVE	Invalid pointer
EUNCH	Unable to fork
ENOTHEAVY	E's my brother
EIMPOTENT	Unable to fork
EMORTAL	Can't kill process
ELABORATE	Too tense
EIA	Standardization violation
ECRIFT	Bit rot in program
E<unprintable>	Obscene kludge in program
ENOUGH	Time limit expired
EEEEEEEEE	Infinite loop in program
EUUG	Intercontinental junket

Lots of network and distributed files systems stuff today. Greg Chesson, chairing one of the Distrib FS sessions coins the term *Doofers*. This is an immediate winner. We immediately have people talking about *rufus*, *goofers*, *newfus*, and so on. Andy Rifkin does his Rufus talk. He's been to charm school recently, and almost makes it sound plausible.

11. Thursday PM

Good lunch with the guys from Lachmann. Feel I ought to be present at the paper given jointly by them and us. Mike Wilde does a great job.

Ronald Hughes' doofer is called *truefus*!

Tea, and another 50-60 entries in the *errno* box. They are coming in thick and fast now. After tea, check out the mail session. Craig Partridge, of MMDF fame, elaborates plans for domain based mailing everywhere.

An interesting scheme for a /mail file system under the V8 FSS. Interesting, but impractical. Nice user interface though.

12. The Conference Dinner

Got talking to a guy working for AT&T who used to work in the UK. Offers me a lift to the conference dinner. We get lost, and drive around Atlanta for a while, eventually finding the Georgia Railroad Depot, a small, shack like building in the middle of a huge parking lot (I'm really getting to grips with this language).

Inside, things are really going strong. There is plenty of booze, even Heineken Lager which is somewhat better than the American gnats' water. And food! Wow, whole pigs, corn on the cob, mountains of food. The whole place is set up like a fair ground, with side shows, the lot. Won vast numbers of plastic toys. Getting decidedly merry.

The party finishes around 9.30, and it's back to find the hostility, sorry hospitality, suites that are still going. DEC play host, and the European committee (Nigel, Jean, Mike O'Dell, Jaap, DT and me) get down to judging the *errno* contest. There have been some 800 entries. 600 are weeded out immediately. They are too sick, to obvious or too unfunny. Sorry guys. That still leaves 200. After three more passes, each getting harder to throw things out of, we are down to a top 14

EDINGDONG	The daemon is dead
ELECTROLUX	Your code could stand to be cleaned up
ELECTROCUTION	Attempt by finger to reach socket
ENOTONHORSE	Mount failed
ENOPHONEBOOK	Directory does not exist
EEMILYPOST	Wrong fork
ENOCONTEST	The judges decision is final
ENOARMSCONTROL	Silo overflow
EFLAT	File system needs tuning
EGODOT	Endless wait
ECRAY	Program exited before run
EIEIO	Bug bug here, bug bug there
ENOSTRADAMUS	Predicted result
EMRED	A host is a host from Coast to coast, and nobody talks to a host that's close unless the host that isn't close is busy hung or dead

EMRED actually wins it, and must be sung. Special efforts in training singers must now be undertaken. It should be pointed out though, just what MR ED is all about. Mike O'Dell quotes:

Well, the show is about a horse, Mr. Ed, owned by one Wilbur Post, an architect and developer. It is set in some generic California (southern, seems like) location. The central gimmick is that Mr. Ed can talk, but only to Wilbur, who of course can't tell anyone else lest he be placed in the booby hatch. An episode I saw today (reruns) had Wilbur ghosting for Mr. Ed writing his memoirs "Confessions of a Palamino Playboy". Wilbur was ostensibly the author writing under "Mr. Ed" as a pseudonym. The conflict revolved around the book containing some non-fictional facts that clearly happened as portrayed in the book, but that Wilbur could not have known about, but Mr. Ed did.

Finally, my best recollection of the original lyric for the themesong.

A horse is a horse, of course, of course,
and no-one can talk to a horse, of course,
that is of course unless the horse
is the famous Mr. Ed.

(reprise)

People yakkity-yak and speak
and waste your time of day,
but Mr. Ed will never speak
unless he has something to say.

Go right to the source and ask the horse,
he'll give you the answer that you endorse.
he's always on a steady course,
talk to Mr. Ed.

(spoken in character)

"I am Mr. Ed."

For you film bufs, you might will remember a series of B movies with Donald O'Connor and Francis the Talking Mule (whom Mr. Ed claims as a relative) which came out in the late 40's and 50's (a couple even starring an actor now playing as the President of the USA!). The show is clearly a knockoff.

It was a weekly series and ran for several years.

13. Friday

The discussion now is about the announcement of the *errno* contest winners. Should we do a stage presentation of the European entry ENOGLIDER, Pilot fell out? In the end we decide that this is just going to be too complex. Everyone is coached in singing EIEIO and EMRED.

The prizes are obtained during one of the morning sessions: several varieties of Champagne.

The morning session, and more on shared libraries. Some neat stuff on Real-Timing UNIX systems by adding pre-emption to the kernel at critical points. It looks impressive.

Final lunch, then off to the third Operating Systems session. Some interesting stuff on as a virtual machine environment, in other words, using to develop portable operating systems. Nothing too earth shattering though.

14. Time to go home

Finally it is time to leave. What have I learnt? Was it all worth it? Will I get rained on inside the aircraft on the way home? Will I ever remember that my name is not An-thhhh-oww-ny, but Nick?

Well, I can safely say that there was lots to learn, and I managed to pick my way between getting bored stiff attending every session, and having a great time talking to legendary people. Lots of good ideas picked up.

Definitely worth it! (Especially keen on the roast pig!)

I came back on British Airways, and their cabins don't leak.

```
(inset) login: anthony
Password:
login incorrect
login:
```

15. Some of the ERRNO Entries

There were around 800 entries to the *errno* competition. It would take me weeks to type them all in, but here are some of the top 300 or so.

E	2.718...
E2BRUTUS	Init killed by an adopted child
E2MANY	Too many error codes
E<unprintable>	Obscene kludge in program
E423	Addictive overflow
E<unprintable>	Obscene kludge in program
E=MC2+1	Illegal units conversion
E?	DMR system error
EACDC	Wrong type of socket
EAGLE	Disk full
EAR	Please repeat question
EASTER	Autoreboot in 3 days
EASY	Cray doing an infinite loop
EAT	Data file munched
EAT	The Computer is going down
EAT	You have been hacking too long, time to ...
EATME	Cannot mount face
EATME	request bytes
EATT	Legal fees exhausted
EATT	Running Sys 5
EATTABOY	Nice try - try again
EBAD	Bad error (not good). Naughty Naughty.
EBADDOG	Pointer on carpet
EBADMUSIC	Warning - muzak playing
EBADTASTE	Compiling an operating system written in PASCAL
EBANKRUPT	Out of cache
EBAPTIST	Busy hands are happy hands
EBB	Data flow reversal
EBCDIC	Dialect unknown

EBCDIC	Non ascii stream
EBCDIC	Non portable character comparison
E BEFORE I	Invalid syntax
EBSD	Running BSD
EBSD	Your tape will be shipped real soon now, we promise
EBUNNY	Multi-hop access attempt
EC00Y	JCL error
ECHAOTIC	Chaotic or random error
ECHERNOBYL	Broken pipe
ECHERNOBYL	Connection melted
ECHIOPRACT	Disk problem
ECHO	Duplicate argument found
ECIA	You're not allowed to know
ECLAMUP	Shell Quit
ECONF	Too many sessions
ECONSTIPATED	Operation would block
ECRAY	Program exited before run
ECREATE	Missing vowel
ECROSSDRESSING	Violation of strong typing
ECRUFFT	Bit rot in program
ECRUSADE	Religious error (SVRn)
ECT	Addressing, I/O, Wrong Command etc
EDDIE	Thank you for making a simple error-code generator very happy ...
EDFS	Not a tty
EDINGDONG	The daemon is dead
EDMR	A host is a host from coast to coast, and no-one can talk...
EDOOFS	Beating dead horse
EDOOFS	Overload
EDOWHAT	Operation unknown in Texas
EDRANO	Sync failure
EDROUGHT	Stream failed
EE	Stoned EE grad student code found
EE6NONE	Kernel non-existant
EEC	Common Market - no parity

EECHO	Bless you
EECK	Mice in machine room
EECS	Invalid department
EECUMMINGS	Case Translator Finished
EEE	Center of terrorist activity
EEEEEEEEE	Infinite loop in program
EEEEEEEEEEE	Speaker too close to microphone
EEEEEEEEEEEEEEEEEEE	Infinite loop detected
EEK	Dead mouse
EEMILYPOST	Wrong fork
EENYMEENYMINYMOE	Scheduler can't decide which process to run...
EFAULT	Earthquake
EFGHIJ	Long time no C
EFIXED	No children
EFLAT	File system needs tuning
EFLAT	String out of range
EFLOOD	Overflow in stream
EFORK	Routing table full
EFORK	Too many forks
EFREETRADE	Setquota failure
EGAD	Got another dump
EGAD	Surely you jest
EGADS	Sudden realisation were running VMS
EGADS	The system is astounded
EGGONYOURFACE	Memory fault: core dumped
EGOAWAY	System is busy now
EGODOT	Endless wait
EGODS	Too many Baptists
EGOEDEL	Maths Argument Undecided
EGOOD	Good error (everyone should make(1) one)
EGREP	String not found
EH	Canadian user error
EH	Say what?
EHEADACHE	Connection request denied
EHEADACHE	Dual processors not tightly coupled

EHTRAP	Hack ...
EIA	Standardization violation
EIBM	User error. Fix the problem and recompile
EIEIO	Bug bug here, bug bug there
EIEIO	Farms in Berkeley
EIFORGET	Out of memory
EILLEGAL	Illegal error (failable)
EILLITERATE	Can't read or write
EIMPOLITE	Bad fork
EIMPOTENT	Unable to fork
EINF	Infinite loop
EJIHAD	Religious error (BSD4.x)
EKEN	You are not expected to understand this
EKNOCK	Who's there
ELABORATE	Too tense
ELAPSED	Sins undeclared
ELASTICK	Clock needs rewinding
ELAWFUL	Lawful error
ELECTROCUTION	Attempt by finger to reach socket
ELECTROLUX	Your code could stand to be cleaned up
ELF	Tolkein ring passing error
ELICIT	Need more input
ELIM	Outside bounds of window
ELUSIVE	Invalid pointer
EMACS	Program too large
EMAIL	Junk mail detected
EMAIL	User is too big a flamer
EMARCOS	No longer supported
EMEACULPA	System error
EMIRROR	Hardware error - see ESMOKE
EMISC	None of the above
EMISS	Record skipped
EMO	Main processor overworked
EMORDOR	Name server bound
EMORTAL	Can't kill process

EMPOTENT	Pipe too soft
EMRED	A host is a host from coast to coast, and nobody talks...
ENAP	On wrong side of window
ENEEDWINDEX	Unable to do I/O to window
ENIH	There's a better way to do that
ENIXON	Tape problem
ENOAIR	Read on a full pipe
ENOARMSCONTROL	Silo overflow
ENOBOZOS	Stupid request
ENOCH	Race condition
ENOCHICKEN	Serious colonel problem
ENOCONTEST	The judges' decision is final
ENOCORN	Serious kernel problem
ENOCURRENT	Bad socket
ENODICE	Error in rand
ENOENO	Musician not found
ENOENO	Too intense
ENOERROR	Just kidding
ENOFLUID	(Cray 2 only) CPU fluid level low
ENOGOOD	Invalid system error
ENOGREYHOUND	Bus error
ENOH2O	Read on an empty pipe
ENOHOPE	System V
ENOKISS	Buss error
ENOKNOW	Definitely hardware or software error
ENOMAAM	Missing token
ENOMULTIHOP	You should use NFS not RFS
ENON	No errors. Just kidding
ENONCOGNISCENT	Something is wrong and I won't tell you what
ENONO	You can't do that
ENOONE	Both endian
ENOPAPER	Read on an empty file
ENOPHONEBOOK	Directory does not exist
ENORMOUS	Program too large
ENOSALT	Nuke error: core dump

ENOSAUSAGE	Serious link problem
ENOSELECTRIC	Not a typewriter
ENOSTRADAMUS	Predicted result
ENOTATE	Not in art gallery
ENOTCOMPLETE	Dial again or ask your operator for assistance
ENOTHEAVY	E's my brother
ENOTME	Not my fault
ENOTMYFAULT	Some other process generated the error
ENOTONHORSE	Mount failed
ENOUGH	Infinite loop skipped
ENOUGH	No ugly hostnames allowed
ENOUGH	Time limit expired
ENOUGH	Too many errors
ENOUNIX	"But that's not UNIX" (you have invoked a system call...)
ENOVICE	Not in Miami
ENOWARP	Out of dilithium crystals
ENOWAY	No way to list all these error codes
ENSA	No such error
ENUF	No more errors please
EOOEAA	Invalid hex format
EOOPS	Program lost
EORGY	Mount table full
EQUTOFGAS	Bus error
EOW	Stack fell over
EP&p6	String not null terminated
EPDP8	Obsolete CPU
EPLUNGER	Unable to flush
EPPRIVATE	Can't tell you error
EPUBLIC	Everyone knows better
EPUN	That's not funny
EPUNT	Now what?
ERABBIT	Too many hops
ERASE	Race condition
ERATRAP	Female programmer induced bugcheck
EREALSOONNOW	Feature not yet supported

ERELIGION	A 4.2/SV system call has been issued on a SV/4.2 system
ERG	Requires too much effort
ERROR	? (the user will know what this means)
ERROR	Error in perror
ERROR	Error
ESALTII	Deadlock detected
ESARTE	No exit
ESCAPE	Trap failed
ESCHWA	Extended character set unsupported
ESMOKE	Software error - see EMIRROR
ESMOP	Code not written; lazy programmer
ESP	Error coming soon
ESPLASH	Cannot push stream module
ESPOON	Too many forks
ET	Cannot phone home
ET	Phone home
ETATJOES	Commercial implementation error
ETC	Misc. error
ETCOMEH	Tbuf fault. Flushing and returning...
ETEENAGER	Runaway child process
ETERNAL	Attempt to sleep on wchan for which wakeup will never...
ETHER	Try STARLAN instead
ETHEWRONGCHOICE	BSD system call
ETHIOPIA	Out of resources
ETICKET	Failure to stop at signal
ETOOMANY	Too many remote file systems
ETOW	Denver boot (wheel clamp)
ETPHONEHOME	Hang up
ETRAP	Rogue fell into its own trap
ETTY	Is a typewriter
EUNICH	/etc/dev not found
EUNICH	Mount request not possible
EUNUCH	Unable to fork
EUSSR	Usr core destroyed

EUUG	Intercontinental junket
EUUG	Too many groups
EVITA	Don't cry for me Argentina
EWATERGATE	Extreme tape gap
EWHAT	Unknown error
EWOODY	This LAN is your LAN, this LAN is my LAN...
EX	Old version detected
EXORCIST	Cannot remove zombies
EXPORT	Feature restricted outside of the US
EYAWN	Algorithm boring
EYOYO	System about to crash
EYUCK	Are you sure that you wanted to mount that VMS file system...
EYUCK	Pipe backed up
EZ	It would be faster to do operation on paper
EZ	Menu in use
EZBAD	Program erased all source and died
EZMUCH	Funny error (Inexplicable without laughing or giggling)
EZRIDER	Irregular machine cycles
EZZZ	Interrupted sleep
EZZZZ	Process asleep
Ep&P6	(null)

The Manchester Competition

*Peter Collinson
Secretary, EUUG*

1. INTRODUCTION

The traditional competition was run as usual during the Manchester conference. The idea this time was to think up a meaningful phrase to go with a UNIX acronym. There were very many entries written up onto large sheets of paper. I then had the dubious pleasure of transcribing these into my notebook. In the process I expect that I missed a few, here are the ones I got.

adb	A dreadful bug
adb	Another damn bug
adb	Antediluvian bug eater
arp	Any respondants please?
at	Accumulate trivia
att	Another terrible trademark
bc	Brilliant crasher
bc	Bypass core
bsd	Bad syntax deterrent
c++	Cannot add
cat	Compose a tune (at Bellcore)
cb	Crash and burn
cc	Change channels
cc	Core dump generator
cd	Can't define
cd	Concoct data
cdb	Cannot detect bugs
chmod	Can hinder many other developers
chown	Can't have ownership work nohow
chown	Chase owner
ci	Correction impossible
cip	Clip important parts
co	Correction ordered
cpio	Corrupt process in operation
cpio	Pipe construction process in operation
cron	Can run only nightly
cron	Continually repeat obnoxious no-good doers

csh	Can severely hinder
csh	Can't and shan't help
csh	Complete sh
csh	Crash system and halt
csh	Crash system hardware
cu	Cockup
cu	Completely useless
cu	Convert to unary
date	Display a time estimate
dbx	Damn bad 'xperience
dc	Divide and conquer
dc	Dumb crashes
dd	Danish design
dd	Data destruction
dd	Destroy disc
DEC	Disgustingly expensive computersa
ded	Delete everything directly
delta	Delete target file
delta	Destroy every last trace archived
ditroff	Do it read only for fun
doc	Describe our computer
dsw	Disc scribbling wanted
du	Device unavailable
dump	Destroy user's modified programs
dump	Don't use my program
dumpS	Drinking unmoderately makes people SNORE
ed	Eject disc
ed	Extremely devious
EIO	Execute invalid opcode (microcoded machines only)
em	Evacuate memory
emacs	Eats memory and cpu superbly
emacs	Editor managed auxillary command system
emacs	Eventually makes all computers sick
fsck	Failed system call, Ken
fsck (-p)	foolish suggestions can kill (less painfully)

fsdb	For sex, dial Berkeley
gfs	Green fresh sausages
grap	Graphical replicated array processor
graph	Generalised rough analysis presentation heuristic
grep	Get ready to eat the processor
grep	Greatest request ever produced
grep	Guess random element and print
grep	Gurus recovery expert program
hhcp	Hardly hopefully connection process
iso/osi	Inside out/outside in
ln	Lindsay's Nfs (implementation)
lp	Lose printout
lpr	Lost printout repeatedly
lpr -m	Mail me that you've lost printout repeatedly
ls	Last supper
ls	Less sex
ls	Let's see
ls	Long story
ls -l	Lots of silly lines
m4	Slow route to nowhere
mac	Mouse and computer
mail	Make and infinite loop
mail	Mangle and incarcerate protocols
mail	Many annoying interruptions from layabouts
mail	Move applicable info long-distance
make	Make aspiration knotted exhumation
man	Male aid needed
man	Many ambitious notes
man	Minimum assistance to novices
mh	Monster hack
mknod	Yet another distributed file system talk (zzz).
mmdf	Multi machine destruct facility
mmdf	My message didn't find you
modem	Moderate outlay data exchange medium
mv	Make vanish

mv	Make volatile
ned	Never easily done
netnews	Notify every techno nerd every week, slowly
news	No-one ever writes sensibly
nfs	No fear, Sun
nfs	Not fresh sausages
nl	Not listed
notes	Networks only total erasure system
nroff	Nearly random output file formatter
od	Obscures data
od	Oh dear
osi	Ossified system interface
osi	Outsiders slip in
osi	Over simplified for idiots
passwd	Pick a seven symbol word, dummy
ps	Print and smear
ps	Print suspects
ps	Profanely slow
rcs	Random code segment
rcs	Random code shuffler
readnews	Really esoteric articles deleted; nauseating essays were saved
reset	Repair Emacs settings
rfs	Real fresh sausages
rfs	Realistically, far from safe
rm	Read manual
rm	Realise mistake
rm	Reinitialize meter
rm -i	Realise mistake (after impatient response)
rm -r	Read manual — religiously
rmdir	Randomly modify directories in root
rn	Remote network
rpc	Random protocol choice
rsh	Replicate superior hackers (BSD)
rsh	Restrain pSeudo hackers (Sys V)
sccs	Scramble code, crash system

script	Some complete rubbish, I probably typed
sdb	Should do it better
sed	Sex evolves desire
sed	Strips extraneous data
sed	Suffer endless delays
sendmail	Simulate eventual (notional) delivered mail and immediately lose it
sendmail	Smart enigmatic non delivered mail archiver
sendmail	Someone else not doing my interpretation logically
sendmail	Start endless new daemons making all infinitely late
sendmail	Stomp every naive, defenceless mailer article into lumps
sendmail	Sundry elusive notes delivered misaddressed and instantly lost
sh	Shan't help
sh	System halt
shutdown	System hung up today due to our work on NFS
sleep	Snoring loudly, eventually nags people
slip	Should link in properly
sort	Shift, organise, rotate — truncate
su	Suspend user
suntools	Someone upset Newcastle to outhype Lindsay's system
sync	Sorry your network crashed
sync	Success yet no chance
sync	Swedish yacht near Chernobyl
tar	Totally amazing results
tar	Totally arbitrary results
tbl	Trademark, Bell Labs
teco	Type Esc to complete operation
tee	Tried everything else
telnet	This ether Lan needs extra transceivers
termcap	Terminal emulator requires mangled cursor addressing protocol
tip	Transfer incomplete programs
tr	Text rape
tr	Too risky
tr	Tortuous request
troff	Try running our fast formatter
uniq	Uniq neglects iterative quantities

unix	Umpteen new implementations expected
unix	Utterly nauseating in 'xtension
UNoItmaiXense	Think about it!
uucico	You see.... I see..... Oh!
uucp	UNIX under constant pressure
uucp	Useless under protocols
vi	Vacillating irradicator
vi	Very insensitive
vi	Violate intuition
vi	Virtually impossible
vi	Virtually incomprehensible
vi	Virtually intelligible
vi	Visually inept
vnode	Virtually no design
wait	Wow! Another idiot time consumer
whatis	Want help: ARGH this is sick
yacc	Yell at compiler constructor
yacc	Yet another core dump created
yacc	You are completely confused
yacc	Yum, another CPU cycle

The judging committee had a hard time with so many entries. Entries which were thought to be worthy of special mention were:

cc	Can't cope
df	Disc full
emacs	Eight megabytes and continual swapping
time	Totally inaccurate measure of execution
unix	Unlimited new interfaces 'xpected

The winning three were:

sh	Sans histoire
sendmail	Seems every new domain means adding information locally
patch	Please apply this clever hack

Uniforum, January 1987

Philip Peake

Axis Digital

Washington DC

This year, UNIFORUM chose to hold the exhibition and conference in Washington DC, which is somewhat closer to Europe than is Anaheim, which was the site chosen last year. Washington is somewhat more impressive than Los Angeles, but even so, still seems to lack something when compared to European cities.

I arrived there on Sunday evening. There was little to do, since I was tired after the flight, and didn't have much money left out of the 50 dollars I had taken with me (the taxi fare from Dulles airport to Washington is 30 dollars). Next morning, I planned to visit a bank and get some more money using my VISA card.

However, next morning was "Martin Luther King day", which basically means that everything (and in particular, banks) is closed. It was also pouring with rain. I walked to the White House, and was suitably disappointed by its unimpressive size.

I looked into the METRO, which seemed impressive, but since the tickets are sold by machines, and I didn't have the correct coins or notes, I couldn't try using it (and I couldn't change the notes I had, because everything was closed ...).

Washington Conference Center

The conference and exhibition was held in the Washington Conference Center, and a bus was available to take collect people from the hotels and take them directly there. Before going to the center, I found a bank and managed to get some money. They wanted two means of identity before they would give me the money, so I gave them my french driving licence and french ID card (*carte de séjour*), which seemed to confuse them a little, but eventually they gave me the money.

Having missed the first bus, I now took the METRO, which is impressive. However, you can't help the getting the feeling that it is maybe designed more as a public bomb shelter than purely a METRO system.

The first day of UNIFORUM was reserved for tutorial sessions, since these were at extra cost, and not on subjects which seemed particularly interesting. I used the day to register for the conference, and to collect the relevant paperwork (lists of exhibitors, entrance badges, conference proceedings etc.). I then made contact with various other people attending the exhibition.

The Exhibition

The next day the exhibition opened. As usual with UNIFORUM, it was a large exhibition. It took almost all of the three days of the exhibition to visit each of the stands and to see what they had to offer. The overall impression was that there was little new on offer. Although there were one or two vendors who seemed to have heard of Europe, and had started to "Europeanise" their products.

Europeanisation seems to mean that the product will handle 8 bit characters, and that there are various versions of the product with menus and (sometimes) diagnostic message in French or German. If you asked awkward questions about spelling checkers, or hyphenation tables for these things, they were always "Planned to be available in the next release".

Europe was represented by a stand organised by the X/OPEN group. I had the distinct impression that most of the visitors to this stand were, in fact, Europeans. But the name of X/OPEN has apparently been heard in the USA, since during the conference AT&T announced that they were joining the group. The other "rival" standard POSIX, or IEEE 1003.1, also announced during a presentation of their aims, objectives and future directions, that they were to work with X/OPEN in an effort to make the two standards converge, and eventually become a single standard. This is very good news, since it had seemed until recently, that there were going to be at least 3 different standards, SVID, X/OPEN and POSIX. Now it seems possible that there will be only one, which can only be good news for end users of UNIX systems.

Some of the largest stands were: SCO, AT&T, SUN and IBM. The SCO and SUN stands both seemed to be doing good business, but the other two seemed to be staffed by bored looking people who stood around in little groups talking to each other, waiting for people to venture onto their stands.

The Conference

This section should perhaps be called "The Conferences", since there was a complex system of parallel session. Some of the sessions being available only if you had paid the appropriate fee.

My personal opinion is that this could have been much simplified, by refusing to accept some of the presentations, which were little more than sales talks. I found it difficult to know in advance if I was going to hear something interesting, or just a description of someones latest product.

However there were some interesting presentations, amongst which were:

- UNIX utilities in ROM for the HP Integral PC
- MINIX - By Andrew Tannenbaum
- UNIX for real time (HP)

I would recommend finding a copy of the proceedings and reading these. There are others there which are probably interesting according to personal taste.

The MINIX system was perhaps the most talked about item on the timetable. It certainly helped to keep the Prentice Hall stand very busy. They had an IBM PC running MINIX on show, and at the user level it really does seem very much like a V7 UNIX system, and its performance on a floppy based machine is quite acceptable.

The Snow

No account of the 1987 UNIFORUM would be complete without mentioning the snow. It started one morning, with just a light cover. Then it started to snow harder, and it continued all day. By the end of the day 18 inches (40 cm) of snow had fallen. Needless to say, the shuttle bus drivers who were supposed to take us all back to our hotels had gone home a long time before the end of the day's proceedings at the conference center. There were long queues waiting for taxis. The taxi drivers filled their cabs to capacity. Apparently, local law permits them to double their fares when it snows. When I eventually got into a taxi, I (and all the other occupants) found that not only was the fare double, but that rather than it being shared between all the occupants, our driver wanted us *all* to pay double fare. Since it was still snowing hard, there was little choice ...

When it snows, not only do the taxi drivers become mercenary, but all the restaurant staff go home early. That night, Washington was full of pitiful small groups of UNIFORUM attendees wandering the streets looking for somewhere to eat.

In Retrospect

Looking back at the event, the most important items seem to be as follows:

— Europeanisation.

Most American companies now seem to recognise that Europe exists. Even if they don't currently have anything directly relevant to Europe most say that they plan to include us, and our peculiarities such as strange character sets, in their calculations for future products.

— Standards.

The convergence towards a single standard can only be a good thing. The SVID was a good idea. It had the problem of being tied directly to AT&T. X/OPEN was a better bet, being a consortium of manufacturers. POSIX is supposedly completely independent of manufacturers. With AT&T becoming part of the X/OPEN initiative, and the announced convergence of X/OPEN and POSIX there is now a real chance of a universally recognised UNIX standard.

— MINIX.

Andy Tannenbaum has done what so many people have said they were going to do, but never did. He wrote a UNIX look-alike from scratch. Not only that, he has made the source available with fairly minimal restrictions on its use.

It must be recognised that this system is *not* a replacement for a modern UNIX system, but could form the base of a future general purpose OS with most of the functionality of UNIX.

Notes on the Birth of The UNIX Cult

*Peter Collinson (local name)
AS/103/108/121/110 (Galactic designation)*

*Galactic Cult Investigation Team,
Canterbury, Kent
(Small Island off Continent 3, Sol 3)*

Editor's Note

This paper was first presented at USENIX in January 1987.

Notes from some recent archeological findings on the birth of the UNIX cult on Sol 3 are presented. Recently discovered electronic records have shed considerable light on the beginnings of the cult. A sketchy history of the cult is attempted.

Background

The UNIX cult is widespread across the Galaxy now and the surprise discovery of some ancient files in the archives of Intergalactic Brain Machines on Sol 3 triggered the dispatch of an inter-disciplinary investigation team. The files are extremely extensive, occupying all of a small island off the coast of Continent 3. It transpires that the island was taken over by Intergal in the aftermath of the Corporate Wars which plagued Sol 3 some centuries after the birth of the cult.

The team were asked to find out the original meaning of some of the incantations used in UNIX religious practice and also to shed some light on what it all meant at the start.

We should take this opportunity to use the ancient prayer:

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

Earlier versions of this prayer do seem to exist, it is unclear why the form of words altered. "AT&T" was the Corporation where the Creators of the cult worshipped. The Corporation totally disappeared in the wars and many of its original records were either destroyed or altered by the victor in an attempt to "re-write" history. The placement of the country USA on the four continents has been lost.

The Gurus

There seems to be still no trace of the original Creators of the UNIX cult, so we start our examination of the records with a group calling themselves the *Gurus*. The etymology of this word is not quite clear but it does have associations with religious teaching and the High Priests of UNIX are given that title today.

Extract from electronic phone tap of someone nicknamed "dsw", believed to be Daniel Stuart Wilson:

"Of course, we were all isolated in the Version 6 days. When we changed things in the kernel we were on our own. There was no-one to phone to scream for help, in many cases there was no-one on the same site who you

could discuss the problems with. We just had to get down and read that C. Sorting things out yourself was painful but you benefited in the long run. You became a better software engineer. After a bit you became a Guru and could lead others. This is largely bluffing — given a new situation and a knowledge of UNIX you could extrapolate the problems easily and seem to be very clever. It didn't actually matter what you had or had not done, it mattered that you *appeared* to have done a lot and could talk confidently about RK05's, PDP-11/45's, typing *chdir* rather than *cd* and a myriad other things which showed that you were a Guru."

This extract shows clearly that the cult grew in small pockets across the globe. In each centre, a few individuals were given the task of installing the paraphernalia of the UNIX cult and of converting others to its use. In the beginning, it seems that these individuals had little or no contact with each other; curiously, this appears to have strengthened their ability rather than weakening it. Other extracts from the archive indicate that the early practices of the cult were small and simple making it easier for one person to grasp their full meaning. As the cult grew, the practices became more complicated and the understanding harder.

The term *software engineer* refers to a person whose task it was to feed instructions or programs into the primitive versions of the Overlords which were extant at the time. Judging from the emphasis made in the records about the need to generate "correct" programs, this was obviously an artistic task requiring considerable expertise. The reference to "C" implies that there was some official form of speech or perhaps a special religious language which was used to convey instructions.

It is not clear whether the "RK05" and the "PDP-11/45" were the names of the Overlords or some ancillary equipment associated with them. However, the word *cd*¹ meaning "to move from place to place" is still used in some arborial societies on the planet US/115/110/105/120.

The User Group

In current UNIX religious practice, the term *User Group* is used to refer to the congregations at the Hologram Services. It seems that after a while the early practitioners of the cult began to have meetings (where people actually appeared together in the same room). Why these were called "User Group" meetings is unclear, since the early meetings were attended by Gurus and not by Users. At the time, a *User* seems to have been a derogatory term for the un-initiates. However, the head Guru at a site was given the title "Super-User"; perhaps this was to hide the evangelical nature of the Guru's task. The reason for the early meetings was mostly to allow Gurus to inform each other how best to perform religious conversion and how to get the most converts the fastest by *improving the Service to Users*.² As the word spread, the meetings grew in size, expense and quality of surroundings. They proved to be an exceptionally good way of upgrading low level Novices into higher level Gurus. This is a reflection on the early religious books which were aimed at Gurus and were often way above the head of the Novices. Novices were encouraged to attend the meetings to gain by word of mouth what they could not gain from the literature. At these later meetings, the nuts and bolts of UNIX practice were still discussed because many of the Novices were either acting in a support role to Gurus or planning to become evangelists themselves. After a

1. Pronounced *see-dee*

2. This is a contemporary term.

while, the Gurus got bored with discussing *inodes*³ and other topics began to creep in. The meetings gradually altered in character, with the Gurus attending because they wished to see the other Gurus; and more and more Novices attending in the vain hope that they would learn something. The cult had spread so far by now that it was profitable for Corporations to become involved in the selling of UNIX paraphernalia and religious goods. Initially, these *Products* were advertised widely at the meetings. However, the Corporations often sent attendees with prepared scripts who were sometimes not even Users and who had little or no knowledge of intricacies of UNIX practice. The Gurus and Novices were dismayed. On their side, Corporations began to see that the word which they were spreading was falling on somewhat stony ground. This gave rise to the Great Split with a rival organisation being set up primarily aimed at selling and the original User Group concerning itself more with the cerebral activities of the cult. In the end, this was a good thing because the Gurus were able to start deriving benefit from the meetings again since there were now spare slots for educational talks. In fact, the rival organisation was wiped out in the Corporate wars because it had allied itself strongly with AT&T.

The User Group, then, provided some important functions. It supplied a forum for discussion of the practices of the cult. It provided a meeting place for the widespread Gurus who initially met to discuss their work but as time passed they went just to meet each other. It spread the word to Novices; and as the cult grew, it provided a place where new ideas on the direction which things should take could be discussed.

Another more hygienic method of worldwide communication grew out of the cult with the formation of the Network. This is discussed in the next section.

The Communicators

When the cult had grown to worldwide proportions, we begin to see the emergence of a global communication system — the "Network" or "Net". The activities on the Network can be deduced from the electronic archives but the material is so vast that scanning it for relevant information is proving difficult.

The majority of traffic on the network seems to have been communication between Overlords describing various error conditions. Here is a sample:

<*Various repeated unintelligible lines*>

From: MAILER-DAEMON (Mail Delivery Subsystem)
To: <uucp@a4los.uucp>
Subject: Returned mail: Service unavailable

— Transcript of session follows —

>>> DATA
<<< 554 sendall: too many hops (30 max)
554 <megd>... Service unavailable: Bad file number

— Unsent message follows —

<*More of the same*>

There are very many other examples of the same type of message. However, these messages do place things in context — we now know that the communication system

3. The meaning of this is totally lost.

was called *Mail* and from this we infer that the mechanism was intended to permit communication between people. In amongst all the Overlord messages we do find some files which appear to emanate from one person and be addressed to another. A high proportion of the sampled files were intended to probe the capabilities of the Network, often provoking an Overlord error message. We know this because the destination address is the same as the source; in some cases the subject is "Just testing" or something akin to that. This technique was perhaps used to investigate the ability of the Network to transfer files. We are left with a much smaller number of what might be termed "useful" messages. In many cases, these consist of personal trivia showing that the Network was supplying the useful social function of allowing people to make and maintain contact.

Other messages consist of hieroglyphics containing many braces "{}" and brackets "()". This type of message shows some form of regularity in structure but syntax analysis is hampered by the presence of so many exceptions. It is possible that these messages contain portions of religious ceremonial or it has been suggested that this is the ritual language "C" and the exceptions are what were called "bugs". This last suggestion (by a post graduate student on the team) has been greeted with a certain amount of derision.

Mail messages have a recognisable format and can be distinguished from another type of message which occurs in much greater volume. It appears that these messages are part of a system called "The News" broadcast to many sites. The News permitted individuals to send a single message to many people across the world. A sample of the contents imply that the News took over as the main method of spreading the UNIX word when the User Group meetings ceased to be totally useful in this role.

Judging from the beginning of many News files, the News was split into many separate subject headings. Over large periods of time, we see the subjects come and go with abrupt changes in title occurring from time to time. The subjects do not seem to be confined to discussion of topics with direct relevance to the cult. There are many headings which just carry "talk" on various subjects. An analysis of the topics is being prepared as a background document since they fall outside the remit of the investigation.

It might be thought that the News would provide an excellent vehicle for the Super Gurus who must have existed by this time — but far from it — very few messages contain what might be termed confident information. Most seem to carry opinion which is contradicted in later messages. The evidence here is that most authors were Novices. The Gurus had either lost interest in spreading the word, or were simply too busy to wade through what must have been daily oceans of verbiage. It is also possible that Gurus just used the Mail to communicate, perhaps not wishing to impose their definitive opinions on the discussion with the thought that discussion is a healthy academic tool.

The Network spread slowly over most of Sol 3 with some areas being exempt because either they resisted conversion to the UNIX cult or were deliberately omitted for ideological reasons. A single News message from "kremvax", an otherwise silent site, seems to have been met with a storm of protest. This shows that the Network crossed political boundaries and proves the contention that Sol 3 was split into separate economic entities predating the rise of the Corporations.

The Vendors

The Corporations who were involved in the propagation of UNIX paraphernalia and religious goods were known as *Vendors*. The Vendors had a different view of the world from the Gurus, and this difference lead to many schisms in the cult. The

Vendors and Gurus tried to maintain a separate identity at all times. For example, when the Vendors attended the User Group meetings they provocatively wore different religious vestment from the Gurus. They did this to make it easy for other Vendors to identify them and to allow their easy differentiation from the Gurus and Novices.

At the centre of the clash of opinion was a fundamental difference in the perception of "the User". The Vendors were always complaining that UNIX religious practices were not "User friendly". By this, they meant that the act of worship was hard to learn, and the rituals were cryptic. They wished their Users to only deal in simple concepts and never to learn more than the basics of the rituals. The Gurus objected to this view because they believed that the learning of cryptic rituals allowed the worship to proceed faster and that limited exposure to only the simple concepts restricted the Users in a way which was not desirable. The Vendors believed that the Users were fundamentally stupid and without any hope of redemption; and the Gurus believed that the Users were fundamentally stupid but might be saved given the correct tuition.

The early Vendors were peopled by Gurus who had often to undergo the necessary clothing transformation⁴ to demonstrate that they had switched camps. These Vendors were responsible for the spreading of the cult to a much wider and naive User population. Some of the Vendors tried to set up their own breakaway cults to avoid the central control imposed by AT&T, the Corporation where the Creators worshipped. These attempts failed. Other Vendors created Sects which specialised in worshipping the many minor Overlords which had appeared about this time. These Sects were partially successful and converted Users to their way of thinking. The Sects created by this method had special names related syntactically to the word UNIX. Only XENIX has survived as an example of this.

Then seeing all this activity, AT&T decided to become a Vendor.

The Gurus were horrified when the Marketing Staff, regarded today as the front line fighting force of all Corporations, were put in charge of the development and promulgation of the cult. The Creators were no longer allowed to directly influence the development, instead they were to pass their ideas to the Marketeers who would decide what was acceptable or not. The Marketeers spent a lot of time producing a new religious tome to help to guide the developers. The book was known as the *Svid* and laid out the central tenets of the practice. The Marketeers were determined that the Svid should be elevated to the level of all the other religious books. To this end they tried to make the Svid as cryptic as all the others, and succeeded.

After some initial shock, the Vendors accepted the central control which AT&T imposed because they saw that it made their Products accessible to more Users. "Consider it Standard" became the watchword in a Crusade designed to eliminate the undesirable Sects which wished to differ from the Svid.

The Users of the Vendors' Products were certainly pleased with AT&T's decision to become a Vendor. It meant that they were no longer tied to the Products of one particular Vendor but could pick and choose without altering their worship. The Gurus were less pleased until they realised that AT&T were unable to change the cult to eliminate them. Every Overlord where the UNIX cult was practiced still

4. Often referred to as my tie neck. This is inexplicable at present.

required a Guru to perform essential tasks.

The Sects

The UNIX cult was always noted for its propensity to split into separate Sects. In the early days, the Creators had a release policy which made sure that all cult members possessed all the relevant facts in order to fully comprehend the implications of the worship. It was said that Users aspiring to be Gurus needed all the facts because the religious books were written for Gurus, and Users could make no sense of them. Unfortunately, access to the information was much abused because the Gurus immediately used the knowledge to alter things and many minor and major Sects of the Cult sprang into life. The ability of each site to generate its own Sect was somewhat curtailed by the cunning ploy of re-issuing the rituals and practices in a slightly altered form from time to time. The Gurus were soon tired of altering the same things every time a new set of rituals came though the door and they began to leave things alone. Also, by this time, the Vendors had made an appearance and because they believed in the Svid Crusade, they stuck with the orthodox mainstream AT&T view of the cult.

Even so, at the end of the period being researched there seems to have been two Sects with differing practices and rituals. The main rival to the orthodox view was a Guru-lead Sect called the "Berkeley System Devotees". This had sprung into existence in the early days of the cult, taking advantage of the knowledge imparted to them by the Creators. However, the early Berkeley Gurus cleverly distributed their rituals and practices in the much the same way as the Creators and this ensured a wider following. The prominence was noted by a higher power and they were chosen to master the revisions of the Cult which were needed to permit worship on some new Overlords.

These new Overlords had managed to conquer the restrictive memory sizes which plagued many of their forbears. It was said by many that the new Overlords had not got this right but at least they did it. As a result the new Overlords had the potential to allow bigger and more expansive practices. The Berkeley Gurus grasped this opportunity with the objective of creating "The Perfect Ritual"⁵. The Perfect Ritual was defined as one where all the letters of the alphabet were used as a "parameter" specifying a distinct action or phase in the worship. The Berkeley Gurus never managed to create the Perfect Ritual, but they came close.

The Berkeley Gurus distributed their rituals and these became popular because the Marketeers inside AT&T were so busy creating the Svid that they failed to notice that Berkeley practices were slowly being adopted on all the new bigger Overlords. The Berkeley rituals were also liked by Gurus because they were nearer the "Old Religion" laid down by the Creators. In a fit of pique, the AT&T Marketeers decided that they would no longer support the new Overlords and branded the Berkeley Gurus as heretics.

As heretics, the Berkeley Gurus decided to go one step further in altering their Sect. They proposed and executed a fundamental change of direction which was to become a "De-facto Standard". The new Sect was revolutionary because it allowed Overlords to talk to each other, but to do this a whole new litany had to be created. New words entered the vocabulary and new concepts were introduced. For many, worship in the new Sect was slow and unwieldy in comparison to what had

5. The rival Sects referred to this as "creeping featurism"; the precise meaning of this obscure phrase is under investigation.

gone before. But the new practices meant the easy ability to interconnect Overlords and this was demanded by the Overlords themselves.

Unfortunately for the participants in the Svid Crusade, many Users actually insisted on being able to use some of the Berkeley rituals. The pressure from the Users was such that we begin to see Vendors announcing their Products as being "with Berkeley enhancements". Finally, the AT&T Marketeers were forced to incorporate some features of the rituals which did not conflict with the teachings in the Svid. At this time, we also begin to see specially created Overlords which could be used to worship in the practices of either Sect simultaneously, this was known as the "Universe Concept".

The Berkeley Gurus were so broken by the gestation of the new Sect that many left, some to worship the sun and some to seek salvation in the noble task of Pixel Creation. It was thought that the new Sect would be the last to sally forth from hallowed halls of Berkeley because the staff were demoralised and without joy. The Svid Crusaders were pleased, "All we have to do is wait and do nothing", they said. Since they weren't noted for doing much anyway, this wasn't difficult.

However, much to the dismay of the Crusaders, many Novices amongst Berkeley group were promoted to Gurus, and these new Gurus worked to consolidate and strengthen the new rituals. The label of "slow and unwieldy" was not to be applied again. Time and motion studies were performed on the rituals for the first time in the recorded history of the UNIX cult and a little more than lip service was paid to the notion of efficiency of worship.

The Standardisers

One way of defeating the degeneration of the pure UNIX cult into Sects was by the creation of a "Standard". As we have seen, the Svid is one example of this. However, the Svid differed from other Standards because only the AT&T marketeers were in a position to generate a Standard without reference to anyone else. They seemed especially keen that no taint of the Berkeley heresy should appear in their work and so did not consult the Users.

In order that the Svid could qualify as a Standard, the AT&T Marketeers had to promise that it would not alter. They agreed to this because they were determined to ensure that the Svid gained religious significance. This was a good thing for other Vendors who were treating the Svid as a Standard but it is possible that the slavish adherence was detrimental to the fortunes of AT&T in the long run because they were unable to update the rituals and practices to keep up with the demand for change created by the Users.

All the other standards were either created by groups of Vendors working together and ignoring the Users; or by groups of Users working together and ignoring the Vendors. Gurus were rarely involved; they were either too busy and important to sit on committees or just plainly could not see the need for conformity.

The Standards rarely reflected the UNIX practices and rituals which were in use at the time of their creation. All of them seemed to have a speculative element, as if the Standardisers themselves tried to develop or perhaps rationalise the rituals in some way. As a result, the Standards were never standard.

However, the various Standards did give the Users an idea of what was expected of them if they desired to move from one Sect to another. Users who did this often, the so-called *Portables*, learned to use the minimum of ritual and to localise the Sect dependent areas of their worship.

The Portables might have been helped by a Standard for the religious language, C. They were surprised to learn that Standard C was, in fact, a different language from the original and almost no-one had an Overlord which could understand it. The changes were no doubt desirable but came from treating C as a "high-level language" rather than using it in the way which the Creators intended, a method of communicating intimately with the Overlords.

Conclusion

The archives are still being searched for other interesting material but enough has been found to demonstrate the fervent activity which followed the creation of the UNIX cult. We have found no trace of the Creators and barely a hint of the disciples who followed them inside AT&T. We hope that more research may provide some answers and respectfully ask for more funding.

④ Peter Collinson is really the Head of the UNIX Support Group at the University of Kent, Canterbury, UK. He has been involved with UNIX systems since 1976, when in the immortal words of Nigel Martin "UNIX changed me from a Lecturer in Computer Science into a Junior Computer Operator". At Kent, he has been responsible for the writing of Cambridge Ring networking software on the VAX, starting with 32V and continuing in the Berkeley tradition ever since. He has passed through the stage of thinking that UNIX is everything and is now able to recognise the nice things about other operating systems and the nasty things about UNIX. He still thinks that the word UNIX should be allowed to be a noun.

He has not published much in the computer journals but has had many pages printed in the various UNIX newsletters across the world. Most of these words were reporting the activities of the EUUG of which he was a past Chairman and is now a Committee member. Some of the words were a paper "On the Design of the UNIX Operating System" (*login*, July 1984) which he refers to as "The Typing Paper" and did seem to manage to raise a few laughs across the UNIX world.

He still cannot work out whether he should use `***argv` or `**++argv` or `++**argv`. `getopt()` is for Users.

The X/OPEN show in Luxembourg

*Theo de Ridder
ridder@honhio.uucp*

On 27th February the X/OPEN group presented a demonstration of portability in one of the buildings of the Commission of the European Communities (CEC) in Luxembourg.

I went there as a delegate of the NLUUG, but also to witness the beginning of a new epoch. It was a one-day trip with mini-airplanes to see a mini-demo with maximal side-effects. Surely, the commitment of eleven major companies (BULL, ERICSSON, ICL, NIXDORF, OLIVETTI, PHILIPS, SIEMENS, AT&T, HEWLETT-PACKARD, UNISYS) towards a single standard is a fundamental breakthrough and establishes a critical mass for a succesfull UNIX explosion in the coming years.

The presentation of the program was in professional hands. So there was a real presentator, supported by a complete light, sound and video installation. The demo itself, compiling and running the 20/20 spreadsheet package, was done by the software house ACE from Amsterdam.

Having too much experience, I was kind enough not to inspect any machine or any source. Just by looking at the public video-screen I tried to make my conclusions. And it was rather embarrassing to see portibility advertised by transferring source-code on 5½" floppies, to see steps of a makefile without any usage of archive or SCCS facilities, and to see a rather suspicious -DXOPEN flag in any C compilation. Furthermore the idea that portibility is just the transfer and compilation of source-code was wrong. The amount of work to get the result running is much more relevant, and there was no word about that issue.

The problem of any standard is that it is based on the past and not on the future. Still missing are important issues like networking, windowing and graphics. However, there is a second edition now of the X/OPEN Portability Guide in a much more practical shape than the previous one. It consists of five volumes covering commands and utilities, system calls and libraries, supplementary definitions, programming languages, and data management. Rather unusual for a UNIX environment is the inclusion of COBOL, ISAM and SQL.

The X/OPEN group gave up its pure European identity, so maybe we are moving towards a single world standard unifying SVID + POSIX + X/OPEN. Whatever happens, there remains a lot of hard work to be done.

The CV Macros

Neil Todd

Imperial Software Technology

Introducing a new *portable* macro package for troff-like processors

Tired of messing around with your CV, trying to get it just right?

Naked troff getting you down?

Need to change it at short notice to fit the Job Ad?

Yes??

Well, in that case you need THE CV MACROS!!

A simple-to-use troff macro package that takes care of all those messy formatting problems and gives you some added features:

- Optional invisible typeface (.I) that utilises the little known feature of most laser printers that allows the selection of a *white* toner cartridge. This specially formulated toner is completely invisible on white paper, so should your manager see it being printed he'll think you've just screwed up again. But get it home, pop it in the oven at regulo 6 for 15 mins and hey presto! the toner turns black. You never have to be furtive in the print room again.
- A hype typeface (.HY) that selects a special toner that absorbs 24 in every 25 photons that strike it - resulting in a typeface that flashes twice a second under normal 50Hz lighting. A slightly increased flash rate is automatically selected when applying for North American jobs (to match the increased tempo of life there).

CV Health Warning: Due to the energy absorbing nature of this typeface, prolonged exposure to bright lights may seriously damage your job prospects.

- A little *white lie* macro (.WL) sets the following text in a radioactive typeface that has a half-life (in days) selectable via an argument. As the toner decays the text becomes rapidly unreadable.
- A *photocopy glitch* macro (.PG) is available that causes the typeface to be altered by a random dither algorithm applied to the laser beam drive logic. Useful for those embarrassing A level or degree results.
- The *salary* macro (.SL) is available to generate suitable *current salary* and *required salary* lines. This is a complicated macro, but illustrates a number of the lesser known troff features, including .ra (the random number generator). This macro also takes an optional numerical argument, a weekly auto-increment for the required salary. This saves the interview weary job-hunter from editing the CV merely to ask for more money.

The features of this package are open to one and all. So long as you can use troff or similar you've got it made. To avoid excessive use of these novel features (these typefaces are expensive you know!) troff will only function with the **-mcv** macros if the input file name is \$HOME/cv.

Oh, and one last thing, remember to de-crypt it first!

GKS in C++

*John E. Richards
Richards@uk.ac.bristol.qgb*

Praxis Systems Plc

Aspects of a binding of GKS, the ISO standard for 2D graphics, to the C++ programming language are presented. The binding makes use of classes and derived classes to define GKS concepts such as segments and workstations. Operator overloading is used for some GKS functions.

Introduction

GKS [1] is defined in a language-independent manner. Bindings of GKS to programming languages have to be defined in order that different implementations of GKS in one language present the same interface to the programmer. Work on bindings has been done on several languages [2], in particular, Fortran [3], Pascal [4,5,6], C [7], Algol 68 [8] and Ada [9,10]. Some work has also been done on bindings for non-procedural languages such as Prolog [11,12] and Smalltalk-80 [13].

This paper presents some ideas for a binding in a new programming language, C++ [14]. Although C++ is a descendent of the C programming language [15], it has some novel features which result in a binding considerably different to the proposed C binding [16].

Bindings can be categorised into two types: the radical and the reactionary. The radical binding attempts to take full advantage of the language, using the argument that people choose to program in a language because they find its facilities useful. The reactionary view is that bindings should be as similar to each other as possible, which means in practice that they should all resemble the lowest common denominator, the Fortran binding. The latter approach has the advantage that programmers can move from one language to another without learning a completely new binding. The aspects of the binding presented in this paper tend towards the radical view. As such, it is intended that they should provoke discussion.

There can be no standard language binding unless there is a standard for the language. C is caught in this predicament; until the current standards work is complete the C binding can only have the status of a technical report. A C++ binding would be in a worse situation. C++ is an evolving language; there is no standard, or work in progress on a standard, and the creator of the language has stated [14] that it will change to reflect any changes to C. The main reason for investigating a C++ binding is to examine what novel aspects are introduced by the use of the language. This paper shows that C++ provides some interesting facilities which can be used in a natural way in a graphics programming environment.

C++

The C++ programming language is a superset of the C programming language, with a few minor exceptions. The major enhancement it contains is the concept of *classes*, which provide the programmer with a method of defining new data types. It is possible for the programmer to:-

- specify how a new type can be accessed,

- specify the operations that can be performed on it,
- define operations to be performed when an object of the new type is created (by a *constructor* function),
- define operations to be performed when an object of the new type is destroyed (by a *destructor* function),
- provide user-defined conversions from one type to another,
- define a new type in terms of a previous user-defined type,
- manipulate lists of objects of similar types, with dynamic selection of the appropriate functions.

An example will serve to illustrate a number of these points. The following is a definition of a new type, or class, called *point*.

```
class point {
    float x, y;
public:
    point (float, float);           // constructor
    void add (point);
}
```

A *point* consists of two *float* variables, *x* and *y*. As *x* and *y* are defined above the keyword *public*, they are considered to be *private members* of the class and they cannot be accessed except by means of any functions (the *member* functions), that belong to the class. In this case, there are just two member functions, the constructor function, which takes two *float* values as arguments, and the function called *add* which takes a *point* as an argument. When a new *point* object is defined, the constructor function will be called.

The constructor function can be defined as follows:

```
point::point (float xx, float yy) {
    x = xx;
    y = yy;
};
```

A new *point* object is created by a definition:

```
point centre(0.1, 0.7);
```

The *x* and *y* members of the *centre* *point* will be set to 0.1 and 0.7 respectively. A compilation error would occur if no initial value was specified.

The *add* function is used to add one *point* to another, and is written as follows:

```
void point::add (point p) {
    x += p.x;
    y += p.y;
}
```

This adds the components of the *point* specified as the argument to the components of the *point* which is used to invoke the function. The following code shows how it could be used:

```
point centre(0.1, 0.7);
point offset(0.2, 0.1);
centre.add(offset);
```

After executing this code, *centre.x* would equal 0.3 and *centre.y* would equal 0.8.

C++ also permits overloading of function names. Two different classes can have member functions of the same name, or two functions can be defined with the same name, but take different types or number of arguments. The compiler is responsible for resolving any conflicts. For example, two functions called *print* could be defined: one to take an int and one to take a char* argument. The following sort of code is then possible:

```
print(2);
print("The quick brown fox");
```

Also, operators can be overloaded. This permits the definition of suitable operations for new user-defined types. For example, the addition and subtraction operators could be defined for the class *point*, making it possible to write code like:

```
point p1(0,0), p2(-4,8), p3(2,9);
p1 = p2 + p3;
```

Mapping of GKS Data Types

The data types integer, real and string are mapped to the appropriate C++ types, int, float and char[]. These types are given the names **Gint**, **Gfloat** and **Gchar[]**, to correspond to the names used in the proposed C binding.

The GKS enumeration types are mapped to C++ enum types.

The GKS point type is mapped to a C++ class called **Gpoint**:

```
class Gpoint {
    Gfloat x;
    Gfloat y;
public:
    Gpoint() {}
    ...
};
```

The GKS name types are mapped to the names of objects. This is explained in the following sections.

Objects

A major advantage of C++ is the ability to work in an object-oriented way. There are several suitable candidates for objects in GKS. This binding uses concepts such as workstation, segment, polyline, polymarker index, window, and GKS itself, as classes. The GKS functions are then mapped to C++ functions that operate on objects of these classes.

There are several advantages to this approach:

- there is a well-defined set of operations on each object,
- it is possible to perform a set of operations on a list of heterogeneous objects by using derived classes,
- there is a reduction in the number of distinct subroutine names,
- the programmer is given the opportunity of defining new classes derived from the ones defined in the binding.

The GKS Class and Object

The binding contains the definition of a class called **GKS**. This class contains the items of the GKS state list, together with a set of appropriate GKS functions.

For example, the CLOSE GKS function is a member of the **GKS** class and has the name *close*. The OPEN GKS function is mapped to the constructor function for the

class. Opening GKS is achieved by defining an object of the GKS class. The following code opens GKS by creating an object called *g* and then immediately closes it.

```
GKS g(cerr, bufsize);
g.close();
```

The parameters of the OPEN GKS function are passed to the constructor function, which can be designed to detect an attempt to open GKS when it is already open. It is impossible to close GKS before it has been opened, because the *close* function cannot be called unless there is a GKS object in existence.

Workstations

The **Workstation** base class is defined as a workstation state list plus the set of workstation control functions. Other classes are derived from the base class for each different type of workstation. It is possible to describe the relationship between similar workstations. For example, a class **Tek410x** could be defined as a derived class of **Workstation** to represent the Tektronix 410x series of graphics terminals. Then, classes **Tek4105**, **Tek4107** and **Tek4109** could be defined as derived classes of **Tek410x**. This provides a natural way to construct workstation drivers, where there is often a large amount of code shared between similar devices. Only functions that differ in a family of workstations need to be specified; the remainder are inherited from the parent class.

The OPEN WORKSTATION function is performed by the constructor function in a similar way to OPEN GKS. The following code opens, activates, deactivates and closes a workstation:

```
Tek4107 display(conid, "Tektronix 4107");
display.activate();
display.deactivate();
display.close();
```

The first line creates an object called *display* of the class **Tek4107**. This is the OPEN WORKSTATION call. Subsequently, all workstation functions are invoked by calling member functions of *display*. No integer workstation identifier is passed as an argument to these functions because it is not needed; the workstation is identified by the object name. The choice of name for the object is entirely the responsibility of the programmer.

Output Primitives

A class is defined for each type of output primitive. There is a base class for the output primitives called **OutputPrimitive**. This has four derived classes, **Text**, **CellArray**, **GDP** and **GpointArray**. The latter is used as a base class for **Polyline**, **Polymarker** and **FillArea**. **GpointArray** is defined as follows:

```
class GpointArray {
    Gpoint* point;
    Gint    number;
public:
    ...
};
```

An object of an output primitive class is created by a constructor function when the object is defined. The constructor function takes arguments which are used to allocate sufficient space for the object and to initialise it. For example, an object of the **FillArea** class, called *triangle*, would be allocated with sufficient space to hold 3 points, by:

```
FillArea triangle(3);
```

Having allocated space, values have to be assigned to the triangle. The most natural way of doing this is to make the triangle appear to be an array. This can be done in C++ by redefining the array subscripting operator, [], for the GpointArray class. Values are assigned as follows:

```
triangle[0] = Gpoint(0.1,0.1);
triangle[1] = Gpoint(0.9,0.3);
triangle[2] = Gpoint(3.0,2.1);
```

The most important member function for the output primitive classes is the one that actually draws the object. The obvious approach is to define a function called *draw* for each output primitive. To draw the triangle it would be necessary to say:

```
triangle.draw();
```

As this is the commonest function performed on output primitives, it was decided instead to redefine the function call operator, (), to perform the draw function. Thus, to draw the triangle:

```
triangle();
```

This notation leads to a concise form of programming. There is no need to specify the number of points or an array of points in the function call as that information is already contained in the object. The following code shows how to use Text and Polyline objects.

```
Polyline line(100); // allocate a Polyline
                    // and initialise it
for (int i = 0; i < 100; i++)
    line[i] = Gpoint(i*0.2,sin(i));
// Now allocate and initialise a Text object
Text title(Gpoint(9,-1),"Sine Curve");
// Draw the line and title
line();
title();
```

Attributes Each GKS attribute is defined as a class. An attribute class will typically consist of one or more variables, together with a SET and INQUIRY function for that attribute. For example, the MARKER TYPE attribute class is defined as follows:

```
class Gmkty : public GintAttr {
    Gint v;
public:
    Gerror set(Gint);
    Gerror inq(Gint&);
};
```

Where attributes share common types and member functions, base classes are defined. In the above example, Gmkty is defined to be a derived class of GintAttr, which is a base class for integer attributes. Similarly there is a GfloatAttr base class for real attributes. As is shown below, this enables the programmer to manipulate lists of attributes in a straightforward way.

Attributes that belong to the GKS state list are specified as members of the GKS object. If a GKS object called *g* has been created, MARKER TYPE and POLYLINE INDEX are referred to as *g.markertype* and *g.lineind* respectively, as in the following extract of source code:

```
g.markertype.set(2);
g.markertype.inq(i);
g.lineind.set(i);
```

Attributes stored in the workstation state list are referred to by using the name of the appropriate Workstation object. Some attributes are stored as arrays. For example, POLYLINE BUNDLE number 3 on the *display* workstation is referred to as *display.linerep[3]*. This means that a call such as SET POLYLINE REPRESENTATION, which would be the following in the C binding:

```
gsetlinerep (display, 3, bundle);
```

is considerably different in form in the C++ binding:

```
display.linerep[3].set (bundle);
```

Segments

Segments are difficult to handle in the C++ binding as there is a requirement for a segment to have both an internal and an external name. The natural way to define a segment is as an object of a class, Segment. The CREATE SEGMENT function can be bound to the Segment class constructor function. The following code will create a new segment, called *s*, by calling the constructor function.

```
Segment s; // create a new segment
```

Unfortunately, an external name is needed to permit the storage and retrieval of segments in metafiles, so the approach taken in Rosenthal and Ten Hagen's C binding [7] has been adopted and a segment is allowed to have both an internal and an external name. The external name is only used when creating the segment.

```
Segment s(3); // internal name: s, external name: 3
s.close(); // close it
s.del(); // and delete it
```

Segment attributes are handled similarly to other attributes. For example, SET SEGMENT VISIBILITY is coded as:

```
s.vis.set (GVISIBLE);
```

Each segment attribute also has an inquiry function called *inq*. INQUIRE DETECTABILITY is written as follows:

```
s.det.inq (detect);
```

Overloading of Function Names

Wherever possible, the minimum number of distinct function names is used. For example, GKS, workstations and segments are all closed by a *close* member function. Attributes are always changed by a *set* function and their values interrogated by an *inq* function. The result is a big reduction in the number of function names at the expense of an increase in the number of different data types.

Overloading of Operators

Some functions have been bound by redefining operators. Examples have already been seen above in the treatment of output primitives. Three other examples are COPY SEGMENT TO WORKSTATION, ASSOCIATE SEGMENT WITH WORKSTATION, and DELETE SEGMENT FROM WORKSTATION.

Copying a segment to a workstation is bound to the << operator. This operator, which is the shift left operator by default, is used by C++ as an output operator and so seems appropriate for this function. If *display* is an object of class Workstation, and *seg1* and *seg2* are objects of class Segment, then the following

will copy *seg1* and *seg2* to *display*.

```
display << seg1;
display << seg2;
```

However, it is possible to concatenate these operations by defining the operator in a suitable way. The following code is equivalent to the code above:

```
display << seg1 << seg2;
```

This is considerably terser than the equivalent C binding code:

```
gcopysegws(display,seg1);
gcopysegws(display,seg2);
```

The notational convenience increases with the number of segments that are copied to the workstation in the one statement.

The *+=* operator is redefined for the ASSOCIATE SEGMENT WITH WORKSTATION function, and the *-=* operator for the DELETE SEGMENT FROM WORKSTATION function.

```
display += seg1 += seg2 += seg3;
display -= seg2;
```

The *+=* and *-=* operators need to be defined very carefully. C++ does not provide a way of altering the order of evaluation when an operator is redefined. So,

```
display += seg1 += seg2 += seg3;
```

is evaluated as:

```
display += (seg1 += (seg2 += seg3));
```

Parentheses can be used to enforce the desired result, but this is tedious:

```
((display += seg1) += seg2) += seg3;
```

The solution adopted was to redefine the *+=* operator, when both the lvalue and rvalue are segments, to create a "last-in, first-out" stack and place the segments on it. The operator returns a pointer to the stack. If the *+=* operator is passed a segment as an lvalue and a pointer to a stack as a rvalue, it adds the segment to the stack and returns a new pointer. When the last *+=* or *-=* in the expression is executed, the segments are popped off the stack and the appropriate GKS functions invoked. This ensures that the functions are invoked in the correct order and removes the need for parentheses.

The *-=* operator is handled in an similar way.

Operator overloading is also used for the metafile functions.

Metafiles

Metafiles are defined as derived classes of type *Workstation*. One class is required for input metafiles, *MetaIn*, and one for output metafiles, *MetaOut*. This is because input and output metafiles have different sets of member functions. Metafile items are defined as objects of type *MetaItem*.

Opening a metafile is achieved by defining an object of the appropriate class, as for other OPEN WORKSTATION calls, e.g.

```
MetaOut gksm_out (cout, "GKSM output");
```

The *MetaItem* class contains fields for the item type, item length, and a data record. Once the programmer has placed the appropriate information in the item,

the WRITE ITEM TO GKSM function can be used. The function is mapped to the << operator, using the same paradigm as is used for streams output in C++, e.g.

```
MetaItem item;
    // place values in the item
item.type = HEADER;
item.length = 80;
item.data = "This is a title";
    // write it to the metafile
gksm_out << item;
```

The GET ITEM FROM GKSM function is mapped to a *get* member function of the MetaIn class. READ ITEM FROM GKSM is mapped to the >> operator, by analogy with WRITE ITEM TO GKSM. As the MetaItem class contains the maximum item data record length as one of its members, one of the arguments to READ ITEM FROM GKSM can be eliminated.

The INTERPRET ITEM function is mapped to the overloaded function call operator, (), for objects of class MetaItem. For an example of using metafiles, see Section 14.1.

Heterogeneous Lists

A major advantage of the C++ binding is the ability to use heterogeneous lists. A list can be defined to hold pointers to objects of a base class. The same list can then be used to hold pointers to objects of any derived class. Operations can be applied to objects on the list in a uniform way without the need to know the type of the object that is being manipulated. The meaning of the operation that is being applied depends on the actual type of the object and this is only known at run-time.

For example, it is simple to define a list that can hold integer attributes:

```
struct aplist {
    aplist* next;
    GintAttr* at;
};
```

Any object of the GintAttr class, or any of its derived classes, can be placed on such a list. Then an operation can be performed on each object on the list, e.g. to set each integer attribute to the value 3:

```
// p is a pointer to an aplist
while (p != 0) {
    p->at->set(3);
    p = p->next;
}
```

A more interesting and useful example is a list of output primitives. A similar structure to the integer attribute list can be defined by:

```
struct plist {
    plist* next;
    OutputPrimitive* at;
};
```

Objects of any of the derived classes of OutputPrimitive (e.g. Polyline, FillArea, Text) can be stored on the list and then drawn by stepping through the list.

```
// p is a pointer to a plist
while (p != 0) {
    (*p->op)(); // draw it
    p = p->next;
}
```

In effect, the above piece of code implements a simple graphics display list, or segment store.

Examples

Example 1

Overloading operators can make some programs much shorter. The following C++ code reads and interprets items from a metafile:

```
// declarations omitted
MetaIn gksm_in(cin, "GKSM input");
while (gksm_in.get(item) != EOF) {
    gksm_in >> item;
    item();
}
```

This is considerably shorter than the equivalent code using the C binding:

```
gopenws(gksm_in, cin, "GKSM input");
ggetgksm(gksm_in, result);
while (result->type != EOF) {
    greadgksm(gksm_in, length, record);
    ginterpret(result, record);
    ggetgksm(gksm_in, result);
}
```

Example 2

The ability to derive classes from other classes makes it possible for the programmer to define new output objects. In this simplified example a new type, called **Block**, is defined to represent filled upright rectangles. The only operations permitted on a **Block** are to define a new one, to move it, and to draw it. The class is defined as follows:

```
class Block : public FillArea {
public:
    Block(Gfloat, Gfloat); // constructor
    void move(Gpoint); // move it
    void operator()(); // draw it
};
```

The **Block** class is a derived class of **FillArea**. No data items are declared as they are all inherited from the **FillArea** class. The constructor function takes two arguments, the width and height of the block, and initialises the data.

```
Block::Block(Gfloat w, Gfloat h) : (4)
{
    (*this)[0] = Gpoint(0,0);
    (*this)[1] = Gpoint(w,0);
    (*this)[2] = Gpoint(w,h);
    (*this)[3] = Gpoint(0,h);
}
```

The (4) term on the first line of the constructor function is automatically passed to the `FillArea` constructor function and allocates space for an object with four vertices. The `this` pointer is always available in a member function and points to the object for which the function was invoked.

The `move` function changes all the coordinates by a displacement.

```
void Block::move (Gpoint p)
{
    for (int i=0; i<4; i++)
        (*this)[i] += p;
}
```

The `draw` function, which, in common with other output primitives, is performed by the overloaded function call operator, first sets the interior style to solid and then invokes the function to draw a `FillArea` object.

```
void Block::operator() ()
{
    g.fillintstyle.set(GSOLID);
    FillArea::operator() ();
}
```

Once the code has been written to define the new class, it can be used in the following way.

```
Block box(2.5,3.5);
box.move(Gpoint(1.0,1.5));
box(); // draw the box
```

Objects of the `Block` class can be treated identically to objects of the `FillArea` class.

```
// box can be put on a plist
p->op = &box;
// box will be drawn when
// stepping through list.
while (p != 0) {
    (*p->op)();
    p = p->next;
};
```

Conclusions and Future Work

The production of a binding of GKS in C++ has shown the suitability of the language for writing graphics applications programs. GKS concepts can be translated into classes in C++ in such a manner that the programmer is encouraged to think in an object-oriented way. The use of function and operator overloading results in shorter and easier to write programs. The ability to use heterogeneous lists of objects and the possibility of deriving new types from the basic GKS ones are notable benefits.

The number of distinct GKS function names is reduced to only 31 in the C++ binding, compared to over 210 in the C binding. This is achieved by using overloaded function names, and is largely due to the use of *set* and *inq* member functions for each attribute. However, the number of data types is increased from about 120 in the C binding to over 160.

Slater [5] says that there are several ways of producing a Fortran binding but all reasonable bindings would be structurally similar; whereas for Pascal a number of structurally dissimilar bindings can be devised. In C++, the number of possible bindings is probably greater still. In many ways the design of a binding is a matter of personal taste, and the one presented here may well change drastically to accommodate other people's ideas.

The next step is to test the binding on top of a GKS library [17,18] which is written in Fortran 77 and uses a Fortran binding. This should give indications of the usability of the binding and some ideas for improvements.

Acknowledgements

Dr. Mark Rafter of Warwick University for suggesting the method of handling the ASSOCIATE SEGMENT WITH WORKSTATION and DELETE SEGMENT FROM WORKSTATION operators.

References

- [1] ISO
Graphical Kernel System (GKS) - Functional Description
ISO IS 7942 July 1985
- [2] M. Sparks
Graphics Language Bindings - the What and Why
Computer Graphics Forum Volume 4 Number 4 Pages 387-392 December 1985
- [3] ISO
Graphical Kernel System (GKS) Language Bindings - Part 1: FORTRAN
ISO DP 8651/1 December 1984
- [4] S. Antoy and G. Dettori
Towards GKS Binding to Pascal
Proc. Eurographics 83 Zagreb, Yugoslavia September 1983 Pages 211-214
- [5] M. Slater
GKS in Pascal
Computer Graphics Forum Volume 3 Number 4 Pages 259-267 December 1984
- [6] ISO
Graphical Kernel System (GKS) Language Bindings - Part 2: Pascal ISO DP 8651/2 1985
- [7] D. S. H. Rosenthal and P. ten Hagen
GKS in C
Proc. Eurographics 82 North-Holland Manchester 1982 Pages 359-369
- [8] R. R. Martin and C. Anderson
A Proposal for an ALGOL 68 Binding of GKS
Computer Graphics Forum Volume 4 Number 1 Pages 43-57 January 1985
- [9] M. Mac an Airchinnigh
The Specification and Implementation of GKS Application Software in ADA
Computer Graphics Forum Volume 3 Number 2 Pages 153-167 June 1984

- [10] ISO
Graphical Kernel System (GKS) Language Bindings - Part 3: Ada ISO DP 8651/3 1985
- [11] P. Sykes and R. Krishnamurti
GKS Inquiry Functions within Prolog
Proc. Eurographics 85 Nice, France September 1985 Pages 185-191
- [12] W. Hubner and Z.I. Markov
GKS Based Graphics Programming in PROLOG
Computer Graphics Forum Volume 5 Number 1 Pages 41-50 March 1986
- [13] P. Wisskirchen
Towards Object-Oriented Graphics Standards
Proc. Eurographics 85 Nice, France September 1985 Pages 391-400
- [14] B. Stroustrup
The C++ Programming Language Addison-Wesley 1986
- [15] B. W. Kernighan and D. M. Ritchie
The C Programming Language Prentice-Hall [D 1978]
- [16] ANSI
C Language Binding of GKS (ISO version) ANSI X3H34/83-12R5 (Work Item)
January 31 1986
- [17] C.D. Osland
Case Study of GKS Development
SERC Rutherford Appleton Laboratory August 1983
- [18] GKS-UK User Guide Program Library Unit, University of Edinburgh 1986

An NRS processor in C and the future ...

Piete Brooks

University of Cambridge Computer Laboratory

The NRS derived file format is the standard for interchange of information about JANET and PSS names and addresses. A set of interface routines are provided in the portable language, Fortran.

*An alternative (written in C) has been provided which does not provide a direct interface for the users, but produces tables for other utilities (such as MMDF, sendmail, Yorkbox, VMS ...). This is largely described in its own documentation (*A Quick Jog Through Configuring The Nrs Processor*); this document describes future plans, namely a brief description of the proposed FORMAT3, a local NRS server and a proposal to adopt a standard NIFTP suite for UNIX.*

1. The C NRS processor

The current programme will accept format 1 or 2 files (monolithic or as filettes) and produce from these tailorised tables for MMDF, UK-1.4 sendmail (and Reading's smail), Yorkbox (R2.2 and R2.1), VMS coloured book, x25hosts, text, UCL dbextract format DBM files, etc.. It consists of a parser for the tailoring info (e.g. what format, which domains to strip, etc.), the main munger which converts to an internal format and the format specific output routines.

A site will typically process the data twice; once to generate site name lists for the mailer (i.e. picking out the known MAIL hosts and the application relays) and again for the X25 level code to map between names and addresses.

This has been written in such a way that (so far) addition of a new input format or output format have been very minor matters, and porting to non-UNIX systems (VMS and PRIMOS) were also fairly painless. As such, this software is of little interest — it does what is expected of it.

2. FORMAT3

After Salford's minor changes to FORMAT1, they realised that the existing format was not suitable for the extension to ISO and that the communication load on their server is too high, so they are designing a very different FORMAT3 (this has still to go before the steering committee). The two compression methods considered are removing duplicated strings and some form of incremental update. (They appear not to be interested in reducing the load by a factor of four by passing the file through compress) The incremental update involves having each section split into sorted and unsorted parts with the hope that the sorted will be much larger, so that the sequential search through the unsorted part should not slow it down significantly.

The ISO extension consists of additional contexts ftam, jtm, jtm-reg, vt and motis, the new network OSI-CONS, and the registration of NSAPs. The duplicated string removal is helped by splitting names and addresses into sections so that common substrings can be shared (i.e. names are prefix [uk.ac], institution [cam] and entity [cl.jenny] and addresses are DTE, YBTS or CUDF, Application Relay and ISO

information). Gateway information and institution descriptions have also been added.

Together these effectively force an implementor to explode the information to obtain complete strings which can be hashed internally rather than sorted externally.

There are several cases where it would be nice to have some keyed information available which although not essential is highly desirable in the real world, such as this machine should not have multiple transfers in a single call. Is there any simple way to extend the database to include this extra data ? I can image that there may be a lot more black magic when the ISO code appears (such as the information that GECs only accept one transfer per TS connection).

Should the single centralised nameserver, now discarded elsewhere, be replaced by distributed servers ? The obvious unit is the institution, with the current INAMES information available on all servers.

3. Local NRS server

As LANs have become the norm rather than the exception and sites are installing transport service relays between the networks they use, the number of hosts that can use WANs is rapidly increasing. If a host wants to make use of other networks it needs to have a complete and up to date copy of all the tables. The current lookup techniques tend to go for speed of lookup rather than keeping the data compact, so this is likely to burn up a lot of disc space if a distributed filing system is not available.

A solution would be to implement the CHAIR protocol (as defined by Salford) as it is the UK "standard". There are very few implementations available at the moment, so it may be better to implement a very simple ad-hoc connectionless protocol which would effectively map the user lookups into a very simple RPC. Somewhat more reasonable in the Berkeley world, would be to use bind to do the mapping. Some systems (e.g. MMDF) can already make use of bind information. If this could be extended to work on SysV, this would be the obvious candidate.

4. Standard NIFTP

With the omission of a serious standard implementation of NIFTP for UNIX by the JNT, it appears that the user community will have to provide its own. With the final demise of the York contract, the JNT are continuing to throw vast sums of money at Spider in the short term, but they are planning a new scheme to have a manager so that companies wishing to offer UNIX coloured book software can take a standard package and need only write the interface to their own hardware or kernel.

UKNET has repeatedly been able to do this where all others have failed, so if we can get our act together and show the JNT that we can provide such a service, we may be able to get some support. The distribution network is already in place, including a dedicated machine. For the user the joy of this is that a new site can simply tar in a file, run the Configure programme, type make install and have a running system. That is all very well, but not my current prime concern.

I am more concerned about the future existance of a working, up to date suite for UNIX that can be used to make a clean transition to the new ISO implementations which are almost upon us :-). The sites which currently provide UNIX implementations are all having serious staff shortages, particularly in the communications area. Either they will have to freeze their software and live with the bugs, or divert effort from elsewhere. Now would seem the time to consolidate the old NIFTP code into a single source (several implementations already support

many underlying networks and interfaces) and have a UKUUCP style of update moderation. When the active contributers have been found we should be ready to plan the transition to OSI and be able to make considered representation to the JNT rather than having the current unfortunate situation where they have drifted away from their community.

The current candidate (the UCL code) supports sockets (ether, UBC X25, UCL ICPS), the Yorkbox, Camtec's dexpand and the ukc io module (which itself supports DECNET, ...). This is currently running with MMDF and UK-1.4 Sendmail. If this (or an equivalent package) could be supported then a large amount of the wasteful duplication of effort could be avoided.

**EUROPEAN
UNIX SYSTEMS USER GROUP
NEWSLETTER**

*Volume 7
Number 2*

Editorial	1
Packets Vs. Circuits, in Two Centuries	3
Music: a Troff Preprocessor for printing music scores	7
An Overview of the Native Language System	25
Grouse: Messages and Prompts in Programs.....	35
Another Proposal for a News Scheme.....	45
UKUUG 1987 Summer Technical Meeting	47
Glasgow Local UNIX Group.....	49
EUUG	51
Progress of ANSI/ISO C Standardisation	53
X/OPEN — What, Who, Why, When.....	59
EUUG Tape Distributions	61
EUnet.....	67
UNIX Clinic.....	71
Review of POSIX	73

Packets vs. Circuits, in Two Centuries

*Michael Lesk
lesk@cs.ucl.ac.uk*

UNKNOWN AFFILIATION

Should complex transmission systems make routing decisions at each hop of each item, or should routing decisions be made in advance for an entire transmission? Recently this argument has been best known as a dispute between packet switched and virtual circuit computer data networks. In the last century, the railroads had a similar problem: should trains proceed signalman by signalman, or according to complete schedules? If the analogy really holds, heavy traffic networks should prefer virtual circuits.

In the last century, the Midland Railway in England found itself with a problem. Coal wasn't getting through to London. At this time (1890's) they, and all other British railways, scheduled their passenger trains but not their slow freights. A coal train meandered from signal cabin to signal cabin, each signalman deciding whether to send it forward or hold it on a siding. He did that by telegraphing the next signalman forward, and asking whether the line was clear. If the next signalman accepted the train, it moved forward one block section, and then the process repeated.

Readers should understand some of the constraints here. Typical British goods wagons did not then have any kind of brake that could be applied while the train was moving. Until about ten years ago, on British freight trains brakes were often found only on the locomotive, tender and guard's van. Even though these trains were short by some standards (British railways were built with short sidings and locomotives were at that time much less powerful than today), they were still underbraked, and safety was preserved by running at slow speeds. So a coal train out on the line represented a substantial delay to any passenger train caught behind it.

Time-keeping on passenger trains, of course, was important. Signalmen were instructed that delaying any passenger train was a serious error, while it did not matter much how fast coal moved down the line. So they tended to be conservative, and when in doubt to put the freight trains into the sidings. But now consider the consequence: suppose you are a signalman, and all your siding space is full of trains, and an adjacent signalman proposes to send you another train; you must refuse it, because if you can not get one of your sidings empty before it arrives a main line will be blocked.

You can now imagine the consequences: as the traffic builds up, the sidings fill, and once all the sidings at important junctions are full nothing can move. The line may be empty; but the trains will sit there until the sidings clear from the destination back. The Midland Railway eventually had to pay enginemen for a day's work in which their train never moved, and it had to refuse traffic because it had no place to put the trains. Computer scientists will recognize this behavior as "thrashing" if too much working memory is needed by the resident processes the CPU spends its time swapping to disk and only rarely does a process get to run.

The normal English solution to the situation was to put down additional tracks. Two-track railways became four-track or six-track. The Midland didn't have the money for that. Instead, a new general manager named Cecil Paget invented the idea of "train paths". Roughly speaking, the idea was that you could accommodate the irregular freight traffic by having a schedule that displayed not only the trains that ran each day, but also showed the slots for additional trains that only ran occasionally. The schedule showed not just the trains that were, but all the trains that might be.

In a train path schedule, the compiler first works out the maximum capacity of the line. This involves picking minimum intervals between trains, and also deciding on the maximum speed trains can be expected to maintain. Since different classes of trains will run at different speeds, the schedule must present a balance between slow freights, passenger expresses, and everything in between. In addition, both long distance and local services must be provided for. And when trains must meet or pass, the schedule must arrange that this happens at a place with a siding.

Once the full list of possible paths was made, the job of the train dispatcher became easier. Given a new request for a slow coal train to London, he simply looked down the list for the next appropriate path, and telegraphed all the signalmen that a train would run on that path today. They could now send it on confidently because they knew that it would not interfere with more important traffic. At each point where there was a conflict, the schedule would so indicate and there would be a siding. As the train left the mine, its arrival time in London would be known.

Doesn't this all sound familiar? The choice is between pre-routing and dynamic routing. Today we have packets or datagrams instead of trains, we have network nodes instead of signalmen, and we have store-and-forward buffers instead of sidings. But we have the same basic problem: in heavy traffic, dynamic routing can overflow buffers and prevent a guarantee of service. Virtual circuits may instead deny a circuit, but once the circuit is allocated the routing is known and the service dependable. Some papers in the literature recommend virtual circuits for heavy traffic (Butrimenko, 1979); others recommend datagrams for heavy traffic (Matsushita, Yamazaki, and Yoshida, 1977).

What is the lesson? If we accept the analogy, we should conclude that

- a. virtual circuits should be preferred to datagrams in heavy traffic situations;
- b. pre-routing is more efficient than dynamic routing.

Of course, there are differences between railroad scheduling and data switching. The variety of routings in data networks far exceeds that in railroading, and the ratio of data transmitted to storage in the network nodes is greater. But don't be too smug about present day systems: after all, no Victorian railwayman could throw away a train of coal and ask the mineowner to please retransmit.

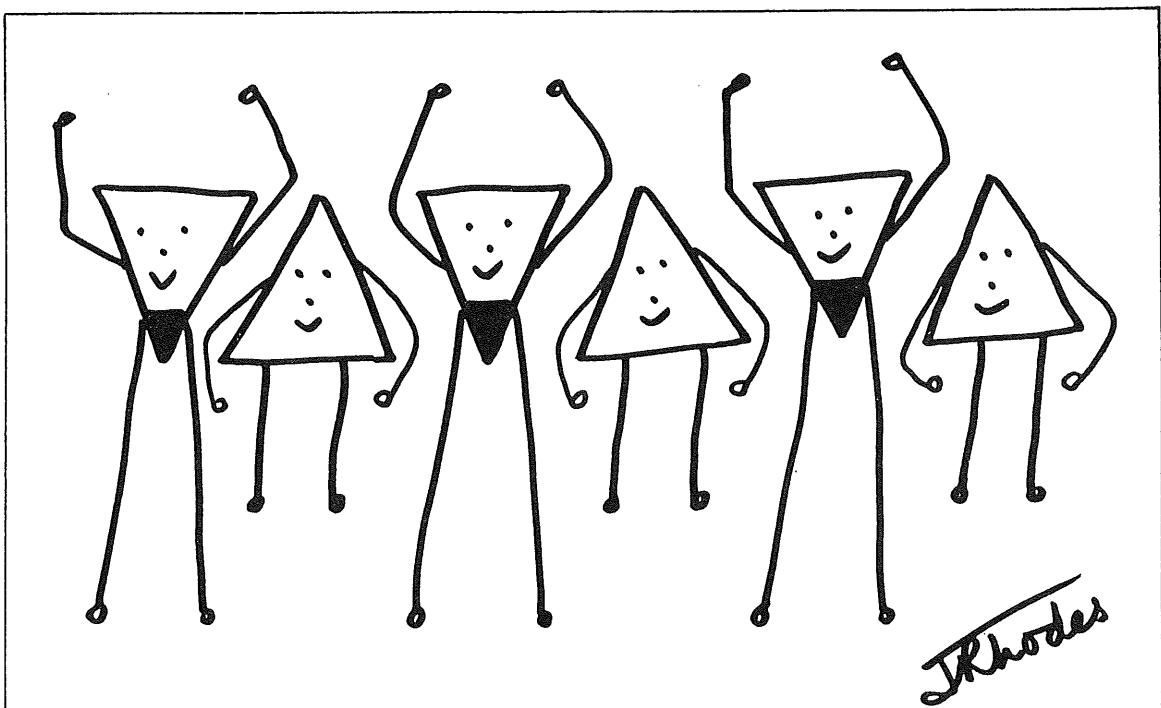
References

- O. S. Nock *Steam Railways in Retrospect* A. & C. Black London, 1966.
 Cecil J. Allen *Railways of Today* F. W. Warne London, 1929.
 F. J. Salzborn *Timetables for a Suburban Rail Transit System* Transportation Science Volume 3, Number 4, Pages 297-316, 1969.
 Yutaka Matsushita, Haruaki Yamazaki, Isamu Yoshida *An Evaluation of Virtual Circuits and Lettergram Services Computer Networks*, Volume 3, Pages 287-294, 1979.

Alexandr Butrimenko, Ulrike Sichra *Virtual Circuits vs. Datagram - Usage of Communication Resources* Performance of Computer Systems, Pages 525-537, North-Holland, 1979.

Vinton G. Cerf, Peter T. Kirstein *Issues in Packet-Network Interconnection* Proc. IEEE, Volume 66, Number 11, Pages 1386-1408, 1978.

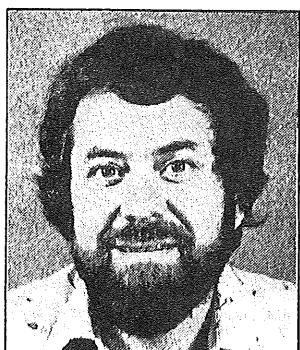
G. Pujolle, O. Spaniol *Modelling and Evaluation of Several Internal Network Services* Performance Evaluation, Volume 1, Pages 212-224, 1981.



UNIX Clones

Music:

A Troff Preprocessor for printing music scores



Eric Foxley
ef@cs.nott.ac.uk

*Departments of Mathematics
and Computer Science
University of Nottingham
U.K.*

ABSTRACT

The *music* preprocessor provides a language for describing music scores, which can then be processed to produce output suitable for the *troff* typesetting system and its other preprocessors, which run under the UNIX† operating system. This document describes the basic facilities available in the *music* preprocessor, and gives examples of its use.

1. Introduction

The output

Cock of the North (*Kevin*)

was created from an input file containing

```
.MS
title = "\fBCock of the North\fP \fI(Kevin)\fP";
timesig = 6 8;
autobeam;
key = a.

e< 'A' d< |
c "A" d c c b a |
a "A" c e f^> "D" e |
\1 |
b "B7" c b b "E7" e d |
\1, 2 |
c> "A" c b "G" g= b |
a>. "A" a> :|
.ME
```

The *music* system is another *troff*^{4,8} preprocessor. It passes most of its input through untouched, but translates those lines between lines ".MS" and ".ME" into commands suitable for the *pic*^{5,6} preprocessor, which can then draw the necessary pictures. Text outside and inside the *music* system can use the full features of the other *troff* preprocessors such as *eqn*³ and *tbl*⁷ if required. A typical UNIX command would then be

```
music source.file | pic | troff -ms
```

or

```
music source.file | pic | eqn | troff -ms
```

if the facilities of *eqn* are required.

The particular rules for layout of the musical symbols are based on examples given in Stone¹⁰ and in the Oxford Concise Dictionary of Music⁹ subject to interpretation and variation. Suggestions for alternative rules would be appreciated, and could perhaps be built into the system as options. For a more general discussion in this area, the reader is referred to¹ for discussion of languages for representing music scores, and to² for information on the current state of computer applications in music printing and in general musicology.

2. The input language

The basic input for the musical score is contained between lines ".MS" and ".ME", and consists of header information describing the output format required, and input defaults, followed by the score details. All text not within a ".MS" to ".ME" section is passed straight through unchanged.

2.1. Header information

The header sets up variables such as the piece title, output width, time signature and key, each of which is specified by an entry of the form

```
<identifier> = <value>
```

The header items are separated by semi-colons, and terminated by a full-stop. A typical header for a straightforward example might be

```
title = "Twinkle twinkle little star";
timesig = 4 4;
key = d.
```

In all straightforward cases a short header is acceptable, since most items default to sensible values. However, the header items have to allow for many variations in output format, and examples of the major possibilities are shown in the following example:

```
title = "...."; # printed at top left of output; the default is to have no title
ctitle = "....";# this title is printed at the top centre of the output, if given
rttitle = "....";# this title is printed at the top right of the output, if given

timesig = 3 4; # sets the default note length to scemi-breve divided by 4,
                # and the default bar length to this times 3
                # the musical length of bars is checked against the bar length

key = g;      # the key of the piece is "G" for both input and output
                # all "F" notes on input now default to F-sharp
keyin = f;    # the input key can be specified distinctly from
keyout = a;   # the output key to produce transposed output
                # the "key =" entry sets both these values
transpose = 1 -1;# instead of using "keyin" and "keyout", this option
```

```

# specifies the number of additional sharps and octave displacement in the
# output key compared with the currently set output key

octave = s; # sets the default octave to that within the stave of the treble clef
# "octave = c;" causes the default octave to start at middle "C"
# "octave = k;" causes the default octave to start at the current key note
# "octave = p;" causes the default octave to make each note as close as
# possible to the previous; the first note follows the "octave = s" rule

bars = 8; # the number of bars in the piece
# used for checking, incorrect value produces a warning
bps = 8; # bars per stave; bars are spaced on the stave proportionally to their
# musical length; notes within a bar are spaced
# proportionally to their musical length
# bps = 0 (default) fits as many bars as possible on the stave

width = 6.5i; # the stave width in inches
# the maximum depends on the output device;
# the current troff default width is 6 inches
# our current laser printer maximum is 7.5 inches portrait
# and 11 inches in landscape mode
height = .25i; # the height of a 5-bar stave
# the default is 0.28 inches
isg = .20i; # increase the spacing to be left between staves,
# the "inter-stave-gap" by 20 units

sig = ckt. # the "c" is "print a clef at the left of each stave"
# the "k" is "print the key on each stave"
# the "t" is "print the time-signature on the first stave"
# the default is to have all

```

The header items are in any order, separated by semi-colons; the last is terminated by a full stop. All unspecified items default from any previous header. An empty header is indicated by a full stop. Further header items will be described later.

Note that all text from any "#" symbol to end-of-line is ignored, and that if the last character of a line is the escape character "\", then the next line is treated as a continuation of the current line.

2.2. Score details

The header is terminated by a full stop, and is followed by the score. The score consists of notes interspersed with bar-lines. There is a warning if the sum of the note lengths in any bar does not add up to the required bar length as deduced from the time signature, or if the total number of bars does not agree with that specified in the header. Both of these checks have been found to be useful.

The pitches of the notes are typed as lower-case letters relative to the current key, as in

d d a a l b b a l g f f l e c d

Sharps and flats of the current key are omitted. Other required accidentals are typed at the first occurrence in the bar using "+" for sharp, "-" for flat, and "=" for natural. For example, "g+" represents g-sharp, "c-=" represents c-flat, and "f=" represents f-natural in a key such as "d". A "+" symbol against an already sharpened note is ignored; a "+" symbol against a note which is sharp in this key, but which occurred with a natural earlier in the bar, cancels the natural accidental. On output, the computer will print only the necessary accidentals, omitting, for example, accidental signs on all but the first occurrence of a given accidental within a bar. A cancelling accidental will be printed if the note is used in the following bar.

The length of a note defaults to the value indicated by the denominator of the time signature, and is thus a quaver if the denominator is 8, a crochet if it is 4, and so on. To specify other lengths, symbols ">" after the note double its length, symbols "<" halve it, "." increases it by 50%, and ".." increases it by 75%. Thus in 4/4 time, "c>." represents a dotted minim, and in 6/8 time "c<<" represents a demi-semi-quaver. As an example, the source

```
.MS
timesig = 4 4;
key = d;
bars = 4.
d d a a l b b a> | g< g< g< g< f. f< l c c d> |
.ME
```

produces the output



The length of the default note for input may be changed by using these note duration symbols in association with the note given as the key in the header. Thus if the key is specified as "d>", the key is "d", and the default note length is double that which would otherwise be expected. If the above example is repeated changing two header entries to halve both the bar-length and the default note-length, the file becomes

```
.MS
timesig = 2 4;
key = d<; # half-length default note
bars = 4.
d d a a l b b a> | g< g< g< g< f. f< l c c d> |
.ME
```

and the resulting output is



For this effect, the key must be set after the time signature in the header, since the "timesig" entry itself resets the default note-length. The full-stop cannot be used to increase the default note-length by 50%, the default can be changed only by factors of 2.

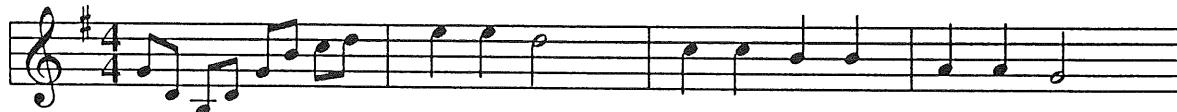
For each note in the source, the letter (and possible accidental) specifying the pitch can be followed by an indication of octave displacement. Symbols "↑" after the pitch indicate an octave upward, symbols "↓" indicate an octave down. The default octave can be set in different ways. It can be set to that from middle C to the B above using the header entry

```
octave = c
```

In this case "c↑↑" is two octaves above middle C, and "b↓" is one note below middle C. Thus in the key of G, the score

```
g< d< b↓< d< g< b< c↑< d↑< l c↑ c↑ d↑> l c↑ c↑ b b l a a g> l
```

produces the output



The octave symbol(s) must at present precede the note duration symbol(s). The default octave can be moved up or down an octave by appending " \uparrow " or " \downarrow " symbols to the note in the key definition in the header. The previous example, if the default key is specified as

`key = g \uparrow ; # g up an octave`

we obtain the output



The header entry

`octave = s;`

sets the default octave to that contained within the treble clef stave, from F above middle C to the E above that. The header entry

`octave = k;`

sets the default octave to be that starting at the current key note. The entry

`octave = p`

causes the octave of each note to be chosen to make its pitch as close as possible to that of the previous note. Thus in this mode the notes

`a b c d | e f g a | a g f c | d c b a |`

would give a run up and down an octave scale. In this mode, the very first note is set according to the "octave = s" rule. Note that although this option minimises the typing of input, it has the unfortunate side effect that octave errors now propagate throughout the rest of the piece.

Bar-lines

Bars are delimited by bar-lines. A limited variety of bar-lines is available, some indicated by an obvious construction, others by a letter following the bar symbol. A selection of these is indicated by

`g>g>|| a>a>|T b>b>|U b>b>|V c \uparrow >c \uparrow >|: d \uparrow >d \uparrow >::: e \uparrow >e \uparrow >:: g>g>|H`

which produces



Note that the system is sufficiently intelligent to replace for example the ":" symbol if it occurs at the end of a stave by a ":" symbol at the end of that stave, and a ":" symbol at the beginning of the next stave.

Rests

Rests are indicated by the underscore symbol " _ ", with lengths specified as for notes using the symbols ">", "<" and ". ". The default length is the same as the value set for notes. Not all rest symbols are currently implemented on the printer. The input

```
a>_>| a _ c.. _<<| a<_<c<_<a _|
```

gives the output



Beams

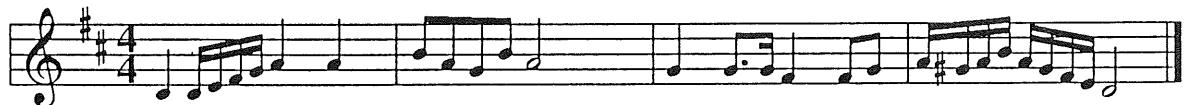
To cause notes shorter than crochets to be joined under a common beam, the notes to be joined are put within square brackets, as in

```
[a b c] [d e f]
```

A beam will correctly fail if it includes any notes longer than or equal to a crochet; at the moment, it also fails if it includes any rests. The latter restriction should be removed soon. A beam cannot cross a bar-line. As an example of beams, the source

```
timesig = 4 4; key = d.  
d↓ [ d↓<< e↓<< f<< g<< ] a a | [ b< a< g< b< ] a> |  
g [ g<. g<< ] f [ f< g< ] | [ a<< g+<< a<< b<< ] [ a<< g<< f<< e↓<< ] d↓ > |T
```

appears as



To simplify the typing of input in straightforward cases, a system for automatically inserting beams is available, and can be invoked by inserting the additional line

```
autobeam;
```

in the music heading. The automatic beaming uses rules from the Oxford Concise Dictionary of Music⁹ which may not always be exactly what is required. However, automatically generated beams will never over-ride beams inserted manually. Having been invoked, auto-beaming can be switched off by the header

```
autobeam = 0;
```

As a further facility, the auto-beamer can be instructed never to let a beam cross for example a "2 times default note-length" interval using

```
autobeam = 2;
```

Different formulae for the slope of the beam are built into the program; formula number 2 can be accessed by the header entry

```
beamslope = 2;
```

Details are the formulae are available from the author. Other formulae could easily be added; suggestions are welcome.

Text

The title of the piece as (and if) given with the keyword *title* in the header information appears at the top left corner of the output. Other titles can be given to appear at the the centre and at the top right-hand corner of the output using the keywords *ctitle* and *rtitle* in the header.

Additional items of text can be given to be positioned relative to any note or bar. For text associated with notes, the text required is contained within quotes to indicate its positioning. Single quotes ('Moderato') indicate text to be positioned above the stave, left justified and starting at the horizontal coordinate of the note; text in double quotes ("Twinkle") indicates text to be positioned below the stave; and text between "@" signs (@3@) indicates text to be positioned close to the note. Text of the last form will be positioned just above the note if it has a downward stick, and *vice versa*. Thus the input

```
g 'Legato' "Twin-" g "-klc" d "twin-" d "-klc" |
c "lit-" [ c< "-tlc" f↑< @3@ c< ] d> 'Fin' "star" |
```

generates the output



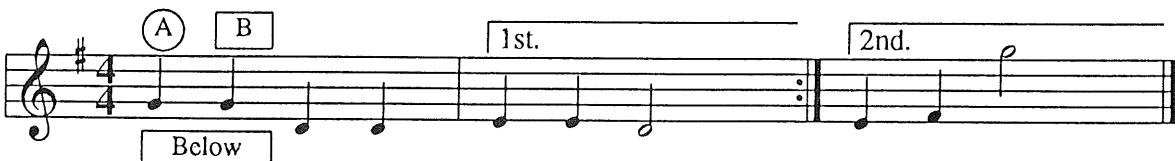
The particular string @3@ is interpreted to imply a triple, three notes in the time of two. The spacing of the notes on the score and the checking of bar-lengths takes this into account.

Text can also be associated with a bar. It is given after the bar-line, and is printed lined up with the start of the bar; it will be above the stave if contained in single quotes, and below if in double quotes.

Text strings starting with the escape character "\\" are treated specially. Any text string starting "\\ b" appears in a box, any starting "\\ c" appears in a circle, and text starting "\\ 1" or "\\ 2" above the stave is used for alternative bars on repeats. The second type, starting "\\ c", will be expected to be single character strings. For example, the file

```
g "\\cA' "\\bBelow" g "\\bB' d d | e "\\1st." c d> :| c "\\2nd." f g↑> IT
```

produces the output



Text starting "\\>" is assumed to be a *diminuendo*; its width will stretch over two notes if written "\\> 2n", over two bars if written "\\> 2b", and over 2 inches if written "\\> 2i".

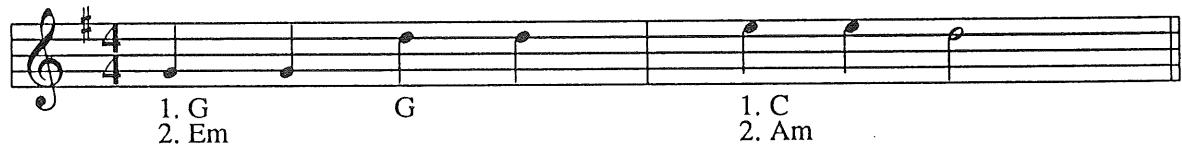
Text between @ symbols starting "\\.", "\\-", "\\<", "\\>" is interpreted as accent symbols and positioned accordingly.

It is hoped that most recognised musical symbols (such as pause and segno) will eventually be included in the standard font; such additional characters are then accessed by the usual methods appropriate to *troff*, such as the "\\(sh" for the sharp-character, and "\\(ft" for the flat-character.

If text is given as two strings, such as

```
"1. G" "2. Em"
```

the two text strings "1. G" and "2. Em" appear one above the other, vertically aligned at the left, as in

**Ties**

Ties are indicated by putting parentheses (round brackets) around the notes to be joined under the tie. Their production is through the use of splines in *pic*; an angular version is available if splines are not provided.

Duplicate copies of earlier bars

The notation

\4|

cause a duplicate of bar number 4* to be inserted at this point. The similar notation

\1,3|

causes bars 1 to 3 to be duplicated at this point. Numbering starts at bar 0 for the lead-in notes, and bar 1 for the first full bar. The notation

\-1|

causes a duplicate of the previous bar to be inserted (current position minus one), and

\-4, -1|

inserts copies of the preceding four bars. The notation

\'Legato'+2, \'Legato'+4

duplicates source from the bar two beyond the most recent which contains the text 'Legato' to the bar four beyond it. Such a bar must occur earlier in the current ".MS" to ".ME" section. Only complete bars can be copied.

It is unwise to use absolute bar numbers in cases where an extra bar could perhaps be added or deleted at a later stage. Relative bar references (using the -4 or 'Legato' notations) are safer in such circumstances. References to bars not yet encountered are unacceptable.

Changes of signature

The notation

\timesig = 3 4.

occurring at the start of a bar resets the time signature, default note length and bar length on the fly to a new value, causing the new value of the time signature to be printed on the stave. The input and/or output keys can be reset similarly using, for example

\key = g<.

resets both input and output key, or

\keyout = d.

resets only the output key. If such a change alters the pitch of the output key, the new key signature will be printed on the stave at that point; if it is used to change only the default note length or octave on input, no

* This bar number must already have been entered, i.e. we must be currently positioned at bar number 5 or beyond.

key signature is printed.

The notation

\barno = 25.

moves the source to the start of bar 25; if the specified bar number is ahead of the current bar, any intervening bars are filled with rests. If the bar number given is less than the current bar, it is assumed that a further part is being added; see multi-part music below for details.

The notation

\sticks = u.

causes the sticks of all following notes to be forced upwards until cancelled by either

\sticks = d.

to send sticks downwards, or

\sticks = x.

to leave sticks free to move in either direction.

The notation

\bps = 4.

resets the current "bars-per-stave" value to the given integer. This can be used to lay out varying specified numbers of bars on each stave. This can however be achieved by a further facility; if the 'l' separator representing a bar-line is replaced by a "!" symbol, the stave is ended at that point.

Any of the above items can be combined as in a normal header, separated by semi-colons and terminated by a full-stop, as in

\bps = 4; keyout = d.

With the exception of "\sticks =", the above notations starting with the escape character "\ are not guaranteed except when used at the start of a bar.

Changes of width

If, say, music is being printed at 8 bars per stave, but there are 4 bars left over at the end, those bars would normally be spread to fit the full width of the page. To reduce their width, use the inserted header

\width = 400 .

which will decrease the width of the page to 4 inches. To reset back to the original width, use

\width = 0.

Source from a separate file

The ".MS" line can be replaced by

.MS < filename

causing music input to be taken from the named file. The file should start and end with ".MS" and ".ME" lines respectively. This enables music source to be easily tested before insertion into the main text. The line can be extended by header items, for example

.MS < filename keyout = b.

Any information given on the ".MS < file" line and following the filename is read after the first heading of the file which is being read. Thus if the named file contains a piece of music which is required to be printed in several keys, or in several different layouts, this can usually be done using extra header items in

this way. The use of the escape character at the end of a line enables several header items to be given, as in

```
.MS < file keyout = g↑; rtitle = "arranged Foxley"; \
bps = 4.
```

which specifies a new output key, right-hand title and bars-per-stave setting.

Output key

The facility to specify the output key independently of the input key takes account of the fact that a natural in one key may become a sharp in another. A piece may be printed in a key other than that in which it is entered by using the "keyout = ..." facility in the heading of the piece. To print a particular piece in two distinct keys, first in the key in which it was input, and then in a second key, store the input

```
.MS
rtitle = "^\nAs input in key of A\nP";
key = a; chords; autobeam; bars = 8.
```

```
c< d< |
c^ "A" a a b< c< | d "E7" b b c< d< |
c "A" a a b< c< | d< "G" c=< b< a< g= c< d< |
c "A" a a b< c< | d "D" b b "E7" c< d< |
c "A" a< c< b "G" g=< b< | a> "A" |
.ME
```

in a file, then use

```
.MS < file
```

to produce the first copy, and

```
.MS < file rtitle = "^\nOutput in key of F\nP"; keyout = f.
```

for a second copy. This then appears as

Glengarry's March

Input given in key of A major

Glengarry's March

Output in key of F

The musical score for "Glengarry's March" is presented in four staves. The first staff begins with a treble clef, a key signature of one flat (F major), and a 4/4 time signature. The second staff begins with a treble clef, a key signature of one flat (F major), and a 4/4 time signature. The third staff begins with a treble clef, a key signature of one sharp (G major), and a 4/4 time signature. The fourth staff begins with a treble clef, a key signature of one flat (F major), and a 4/4 time signature. Chords are labeled under the staves: F, C7, F, E♭, F, B♭, C7, F.

An alternative technique for transposition is to use a simple

key = d;

header, and to follow it with a header entry such as

transpose = 2;

This will cause all output to be printed two sharps up from the output key currently set, following all output key changes during the piece. Negative values, of course, imply less sharps or more flats.

In transpositions, single sharps, naturals and flats are printed correctly relative to the new key. However, double sharps and flats (arising from, for example, an accidental sharp when transposed to a key in which that note is already sharpened) are re-coded to a new note. It should be noted that the input notation does not currently allow double sharps or flats to be specified in the input key. Both these and quarter-tones will be added to the program.

The additional entry

chords;

in the header of the above example causes the system to assume that any text below the stave represents the names of chords, and transposes them correctly; other text is left unchanged. In chord names, it is assumed that the full note is given, for example "F+m" for a chord based on F-sharp, even in a key in which the note of F is by default already sharpened. This appears to be the general convention.

The "chords;" entry also allows the use of "+" to represent sharps and "-" to represent flats in the text of chord names; the "+" and "-" characters will be correctly replaced by "#" and "b" symbols when printed. To print a "+" symbol, escape it with the "\ character.

If the "chords" entry is not given, text is printed exactly as specified, so that a "+" sign in the text appears on the output as a "+" sign; a sharp sign would be printed by the string "\(sh".

3. Multi-staff multi-part music

For multi-staff multi-part music, the music source must specify to which part each note belongs, and the parts must then be linked to particular staves.

3.1. Specification of stave layout

Two additional lines must be included in the header, typically

staves = t b;

parts = 1 u 1 d 2 x;

These indicate

- a) that there are to be two staves, the first in the treble clef, the second in the bass clef; and
- b) that there will be three parts, the first to be printed on stave 1, sticks up, the second on stave 1, sticks down, and the third on stave 2, sticks either way.* The default is

```
staves = t;
parts = 1 x;
```

To allow for more complicated situations, more information can appear optionally between the clef names on the "staves =" line. If a "=" sign appears between two clef characters, as in

```
staves = t = t b = b;
```

the bar-lines of those two staves will be connected on the final output; in this example the first and second staves will be connected, as will be the third and fourth. If the clef letter is followed by a positive or negative integer, the output on that stave will be in a key offset from the basic output key; a positive integer represents a number of additional sharps or less flats, a negative integer represents a number of less sharps or more flats. This facility is essential for scores involving transposing instruments. In addition this positive or negative key offset may be followed by a number of " \uparrow " or " \downarrow " symbols to indicate octave displacements for that stave relative to the current key. Finally, any string in double quotes following the clef letter will appear at the left of that staff in the output; this is used typically to indicate instrumental parts. The example

```
key = g;
staves = t "flute" t "violin" t "clarinet" +2 t "trombone" -2;
```

will give output on four staves, the first two in the key of G (but perhaps changing during the piece), the third always transposed two sharps up (initially in A), and the fourth always transposed two flats down (initially in F). Permitted clef letters are "t" for the treble clef, "b" for the bass clef, "a" for the alto clef, and "n" for the tenor clef.

3.2. Specifying the separate parts

The music source needs additional notation to identify which of the source belongs to each part. Parts may be given bar-by-bar, as in the two-part music

```
\partno = 1. g g d d \partno = 2. d\downarrow d\downarrow b b |
\partno = 1. e e d> \partno = 2. c c b> |
\partno = 1. c c b b \partno = 2. a a g g |
\partno = 1. a a g> \partno = 2. f f d\downarrow> |
```

When a new part is specified, as in "\partno = 2.", the following source is assumed to restart at the beginning of the current bar. Alternatively, the complete source may be specified part by part, as in

```
\partno = 1. g g d d | e e d> | c c b b | a a g> |
\baruno = 1; partno = 2. d\downarrow d\downarrow b b | c c b> | a a g g | f f d\downarrow> |
```

where the "\baruno = 1" indicates a return to (the beginning of) bar number one.

In multi-part scores, when earlier bars are to be copied, the notation

```
\1, 8 |
```

inserts the current part number from bars one to eight at the current point. The notation

```
\1, 8 p 2 |
```

inserts bars one to eight of part 2 into the current part at the current point. To ignore a particular part which you wish to leave in the file for other reasons, specify it as printing on stave number 0, as in

* The notation "\sticks = u" in the source over-rides the default stave settings.

staves = t b;
 parts = 1 u 1 d 0 x 2 x;

to print the first two parts on stave 1, the fourth part on stave 2, and to ignore the third part.

Facilities to refer to parts by names rather than numbers will be added shortly, together with a transposing option to insert transposed or inverted duplicates of earlier bars.

3.3. Example

Typical source to print a multi-part piece, and then to print the first part on its own, and then to print the last part on its own, would be done by storing the basic score in a file as follows:

```
.MS
title = "Complete score";
key = g;
...
bars = 16;

staves = t b; # treble and bass clefs
parts = 1 u 1 d 2 u 2 d. # two parts on each staff

# source for all parts follows
\partno = 1. ... # soprano part inserted here

\partno = 2; barno = 1. ... # alto part inserted here

\partno = 3; barno = 1. ... # tenor part inserted here

\partno = 4; barno = 1; key = gv. ... # bass part defaults an octave down

.ME
```

To print the complete piece as specified in its header (two staves), use

.MS < file

To print the soprano part only, use

```
.MS < file title = "soprano part"; # all settings remain as specified \
staves = t; # one staff, treble clef \
parts = 1 x 0 x 0 x 0 x. # one part, sticks either up or down \
```

To print the bass part, use

```
.MS < file title = "bass part"; \
staves = b; # one staff, bass clef \
parts = 0 x 0 x 0 x 1 x. # one part, sticks either up or down
```

Alternatively the full score could be printed on its own, and then the soprano part could be printed alone in a separate run by altering the "stave" and "parts" entries in the first heading to read

staves = t;
 parts = 1 x 0 x 0 x 0 x;

i.e. to ignore parts 2, 3 and 4.

To print on four separate staves, but with the second stave always transposed to two more sharps than the current output key, and the third to one less, the relevant header entries would be

```
staves = t t+2 b-1 b;
parts = 1 x 2 x 3 x 4 x;
```

4. Error reporting

A number of suspicious constructs in the source file produce warning or error messages. After an error message, the program will terminate. After a warning message, processing proceeds as normal. Error messages include the line number of the suspect line, and a pointer to the position at which the error became detected. Possible errors include invalid characters at any point and beams which are opened but not closed. A warning message is produced if the notes in a bar do not add up to the correct length for that bar, but is suppressed if the bar starts or ends with a non-standard bar-line. Un-necessarily repeated accidentals also produce a warning.

5. Miscellaneous additional features

Text omission

The header line

```
text = o;
```

causes text under the stave (in double quotes) to be suppressed, text over the stave printed. The line

```
text = u;
```

causes the text over the stave (in single quotes) to be suppressed, while

```
text = ou;
```

causes both to be printed, and

```
text;
```

causes both to be suppressed.

Cancelled Accidentals

The header entry

```
natural;
```

causes an appropriate accidental to be placed against a note which was set with an accidental in the preceding bar. The entry

```
natural = 0;
```

cancels the effect.

Inter-stave gap

To increase the vertical spacing between staves, use the header entry

```
isg = 25;
```

for an inter-stave gap of 0.25 inches. To increase in addition the gap between the combined staves in multi-staff music, use

```
isg = 25 10;
```

where the first number is the spacing between groups of stave, and the second between staves in a group.

Setting defaults

The program first reads a file called "*mus_default*" if such a file exists. It must be a file containing only a header, of the form

```
.MS
sig = kt;
octave = s;
rttitle = "Copyright Fred's Music Ltd";
height = 0.35i;
width = 700.
.ME
```

and sets all defaults for the program.

PIC commands

The output of the *music* preprocessor is fed into the *pic* preprocessor. Any text string starting "\p...." is assumed to be a PIC command. The program moves to the appropriate position (above the stave if the string is within single quotes, for example) and plants the given PIC commands. In addition, the program reads a file named "*pic_default*" on start-up, to enable PIC-macros to be used.

6. Further enhancements under consideration

Possible further enhancements include the following.

Font characters

The font characters need to be enhanced, new characters added, and a variety of fonts for music symbols made available. Different available digitised music fonts are being studied. The exact alignment of all symbols awaits the choice of a font. (A variety of fonts for text is already available.)

Varying the height of the stave

Refinement of the ability to vary the height of the stave; at the moment some slight misalignment occurs, but the output is as follows:

Knick-Knack 2

height 0.24 inches, 8 bars/stave

height 0.45 inches, 4 bars/stave

The image displays two sets of musical staves. The top set, labeled 'Knick-Knack 2', consists of three staves, each starting with a treble clef and a key signature of one sharp. The bottom set, labeled 'A', also consists of three staves, starting with a treble clef and a key signature of one sharp. The music is composed of eighth notes and sixteenth-note patterns. To the right of the staves, the text 'default height 0.32 inches, 4 bars/stave' is written.

Text can, of course, be set to any size using *troff* conventions; the above examples show how the default text size is set in proportion to the stave height.

7. Acknowledgments

Thanks are due to many people for their comments at various stages of the development. The author is indebted in particular to Graeme Lunt who looks after the laser printer, and to Dave Brailsford who obtained it for the department. Other colleagues in my department helped at many points, among them Andy Walker, Jim Duckworth and William Shu. Members of the music department are always helpful, and compensate for my lack of musical expertise; they include Ian Bent, John Morchen and Peter Nelson. The testing of the system has been assisted by Toby Bennett of the Genetics Department, who also devised a system for proof-reading my scores by writing a program to play them directly on a BBC micro-computer.

Eric Foxley
April 24, 1987

References

1. John S Gourlay, "A Language for Music Printing," *CACM*, vol. 29, no. 5, p. 37, May 1986.
2. Walter B Hewlett and Eleanor Selfridge-Field, *Directory of Computer Assisted Research in Musicology*, June 1986. Center for Computer Assisted Research in the Humanities, Menlo Park
3. Brian W Kernighan and Lorinda L Cherry, "A System for Typesetting Mathematics," *Comm ACM*, vol. 18, no. 3, pp. 151-157, 1975.

4. Brian W Kernighan, *A TROFF Tutorial*, 1978. Bell Laboratories
5. Brian W Kernighan, "PIC — A Language for Typesetting Graphics," *Software — Practice and Experience*, vol. 12, no. 1, pp. 1-21, 1982.
6. Brian W Kernighan, *PIC — A Graphics Language for Typesetting : User Manual*, March 1982. Bell Laboratories
7. M E Lesk, "Tbl — A Program to Format Tables," *UNIX Programmer's Manual*, vol. 2, January 1979. Section 10
8. Joseph F Osanna, "NROFF/TROFF User's Manual," *UNIX Programmer's Manual 2*, January 1979. Bell Laboratories
9. Percy A Scholes, *The Concise Oxford Dictionary of Music (2nd Ed)*, 1964. Oxford University Press
10. Kurt Stone, *Music Notation in the Twentieth Century*, 1980. W W Norton & Co

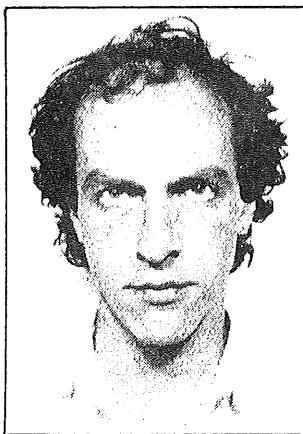
An Overview of the Native Language System

Michael J. C. Terry

mjct@inset.co.uk

...!mcvax!lukcl!inset!mjct

*The Instruction Set Ltd
City House, City Road
London EC1V 9QH*



Michael Terry is a technical consultant at The Instruction Set, involved mainly with relational databases.

He originally studied French and Swedish (with a dash of Norwegian and Finnish) at university, before crossing the great divide and plunging headlong into computing ("because there was nothing left to do").

Having an intimate knowledge of European collating sequences, hospitality, and drinking habits, he is uniquely qualified to be working with NLS. Earlier this year he spent 6 weeks at Hewlett-Packard's Cupertino office working with them on their NLS project and learning how to operate the jacuzzi.

"If English was good enough for Jesus, it's good enough for me."

1. A New UNIX Internationalisation Standard

In January this year, the X/OPEN group† published the second edition of its X/OPEN Portability Guide (XPG)^[1]. Section 3 of the guide included a software internationalisation interface standard specification — the Native Language System (NLS). Although many proprietary solutions to the internationalisation problem have been attempted over the years, this is the first time that a commercial standard has been specified for internationalisation on UNIX® systems.

The X/OPEN NLS standard specification has arrived as a response to a pressure that has been growing slowly but relentlessly from non-English-speaking UNIX users as use of the system has filtered down from the ivory towers of academe to the air-conditioned offices of modern commerce. It is not surprising that this internationalisation specification has emerged from the X/OPEN group rather than from AT&T — after all, despite the recent addition of American companies to the X/OPEN roll call, X/OPEN started out as a purely European grouping, and is still

† AT&T, Bull, DEC, Ericsson, Hewlett-Packard, ICL, Nixdorf, Olivetti, Phillips, Siemens, Unisys

predominantly European. What is perhaps surprising is that the NLS specification is based on an internationalisation architecture developed in the USA by Hewlett-Packard.

Being Europeans, the members of the EUUG will be well aware of the problems that result from the American nature of UNIX — the ASCII codeset does not support many of the characters in the various European alphabets; the system uses US cultural practices (the US assumptions, for example, that the radix character is a dot and that thousands are separated by commas, are reversed in many European countries); moreover, the terse UNIX error messages are all in (sometimes bizarre) English.

The aim of NLS, then, is to provide the specification of an internationalisation framework — a set of utilities and library functions — that enables applications software to be adapted appropriately for use in *any* country or local environment.

2. Major Design Goals of NLS

NLS is designed to provide the following features:

- Support of multiple extended (8-bit) character sets on the same machine, simultaneously.
- Preservation of 8-bit data integrity.
- Multilingual program messages.
- Proper representation of local conventions.
- No need for multiple versions of software for different languages.
- Ability to add new languages without the need to recompile existing software.

In addition, UNIX systems with NLS must still retain the original 7-bit ASCII functionality.

3. Implementing NLS

Hewlett-Packard have a working version of NLS on their HP-UX operating system. The source code has been made available to the other members of X/OPEN in order to expedite its implementation on currently available versions of UNIX. Whether they are using the H-P code or not, the other members of X/OPEN are in the process of implementing NLS on their versions of UNIX.

Thus AT&T, having recently joined the X/OPEN group, are committed to the eventual release of NLS on a future version of UNIX. However, it is unknown if AT&T will ever release NLS on any version of System V.2.

Currently, implementing NLS on a UNIX system entails the creation of new utilities and C library functions, as well as modifications to pre-existing UNIX code. No kernel level changes are required.

The next six sections outline the work involved in implementing NLS on UNIX.

3.1 Extended Character Set Support

Because ASCII is a 7-bit codeset, it is capable of representing a maximum of 128 characters. 8-bit codesets can represent up to 256 characters.

The eventual intention is that NLS will support multiple 8-bit character sets. The XPG states:

This first issue of the X/OPEN NLS specification defines the major transmission codeset for Western European use as the standard IS8859/1, and also recommends its use as the corresponding internal codeset. Other codesets will be identified in later issues.

The IS8859/1 codeset is capable of supporting most major Western European languages. In addition, it is compatible with ASCII functionality, since it incorporates the ASCII codeset as the first 128 characters of the codeset.

3.2 Cleaning Up 8th Bit Usage

Everything sounds hunky dory until we take into account the problem that the UNIX utilities have a bad 8th bit habit — many of them use the 8th bit for their own arcane internal purposes. For example, the shell sets the 8th bit of characters read in from the command line if they were quoted. Consequently, 8-bit data is corrupted if passed through any such offending commands.

Another problem is that sign-extension can occur with 8-bit characters when they are manipulated as integers.

The upshot of this is that the UNIX utilities must be gone through with a fine tooth-comb in order to detect and correct any possible areas of corruption. This is a tedious, long-winded and, in the case of some commands, non-trivial task.

3.3 Character and String Handling

Many languages share the same codeset, but there are differences in the way each language handles the component characters of its alphabet.

Lookup tables must be supplied indicating character class membership, up/downshift relationships, and collation (sorting) orders for each language. These tables must understand 1-2 character mappings such as Spanish *ch* and *ll*.

The affected library routines include the *ctype(3C)*, *conv(3C)* and *string(3C)* routines, which must be amended to make use of these tables.

3.4 Message Catalogues

NLS uses a message catalogue mechanism to provide program messages and prompts in multiple languages. A utility, *gencat(1)*, is used to generate message catalogues from a source file containing program message strings. The library functions *catopen(3C)* and *catclose(3C)* are provided to open and close appropriate message catalogues. The routines *catgets(3C)* and *catgetmsg(3C)* are provided for access to messages from the currently open catalogue.

A message number is associated with each message. This number is used to index into the catalogue to retrieve the message.

An environment variable, *NLSPATH*, can be used to specify a catalogue search path. The advantage of this is that new languages can be added to a system without the need to recompile software. For example:

```
NLSPATH=/usr/lib/nls/%L/%N.cat:/usr/me/cats/%L/%N.cat
```

The special notation *%L* maps to the name of whichever language is currently being used, while *%N* maps to the name of the program being run. Thus if the current

language is *french*, and the command is *rm*, the above maps to:

```
NLSPATH=/usr/lib/nls/french/rm.cat:/usr/me/cats/french/rm.cat
```

If the catalogue cannot be accessed, a default string (specified in the original source code) is printed out. However, if the *catalogue* is opened successfully, but the numbered *message* cannot be found, then a null string, rather than the default, is returned. When such errors occur, this results in strangely silent programs. I think X/OPEN got that one wrong.

3.5 Local Customs Database

A local customs database must be supplied for each language. This contains the names of months and days, currency formats, yes/no strings etc.. A routine, *nl_langinfo(3C)*, is supplied for accessing elements from this database.

3.6 Language Announcement Mechanism

Means must be provided to enable users and programs to determine the language to be used. An environment variable, *LANG*, enables the user to set and reset the language in which prompts and messages from internationalised programs will appear. For example, the following shell commands will force any program that uses the *catopen(3C)* call to access the Italian message catalogues from the directory */usr/lib/nls/italian*:

```
NLSPATH=/usr/lib/nls/%L/%N.cat  
LANG=italian  
export NLSPATH LANG
```

The *nl_init(3C)* routine is provided to set up the working environment for the current language. This routine reads in the appropriate character tables and local customs data:

```
nl_init( "italian" );
```

Nl_init() can be called more than once by a single program, thus for example permitting the same process to work in multiple languages. An example of a very simple program that prompts for a language name and then runs the *date(1)* command in that language is given in Figure 1.

Note that the *%L* notation can be used to force the *catopen(3C)* routine to use the value of the environment variable *LANG* to determine the catalogue to be accessed. The *nl_init(3C)* routine, on the other hand, uses the value of its single argument to decide which character tables and local customs database will be used. Thus it is possible, by setting *LANG* to one language and calling *nl_init(3C)* with a different language as its argument, to have a program putting out prompts in one language (say French) while processing data using the tables for another language (say Norwegian).

```

#include <nl_types.h>
#define NL_SETN 1
char *getenv(), *catgets();

main()
{
    nl_catd nlmsg_fd;
    char newlang[15], envstr[20];

    nlmsg_fd = catopen( "menu", 0 );

    for( ; ; )
    {
        printf( catgets(nlmsg_fd,NL_SETN,1, "Make selection: " ) );
        catclose( nlmsg_fd );
        scanf( "%s", newlang );
        nl_init( newlang );
        sprintf( envstr, "LANG=%s", newlang );
        putenv( envstr );
        nlmsg_fd = catopen( "menu", 0 );
        system( "date" );
    }
}

```

Example output:

```

Select Language: french
lun 04 mai 1987 10h32 42
Choisissez la langue: swedish
Mån 04 Maj 1987 10.32.43
Välja språket: english
Mon. 04 May, 1987 10:32:44 AM

```

Figure 1. Program to Run the date(1) Command in Multiple Languages

4. Handling Mixed Codesets — Software Implications

The nature of the UNIX operating system poses one insurmountable problem when it comes to handling data from a mixture of character sets. UNIX files are merely streams of bytes, and it is impossible to tell what character set the data in a file is composed of.

Although kludges such as file-naming conventions, or storing information on file contents externally to files in some kind of catalogue, etc., have limited usefulness, there is absolutely no way of determining the contents of pipes. The other problem is that a file might contain data from a mixture of character sets.

Consider also a multilingual system where users of different nationalities and different character sets have accounts. The contents of the /etc/passwd file are going to be semi-incomprehensible to everybody. This kind of problem extends to many other areas. In the end, it will probably be necessary to retain ASCII functionality for system administration, or else each system will have to have a basic "nationality" that determines the character set used in system administration.

For the moment, these problems are immaterial, since only a single codeset has been specified thus far.

5. Handling Mixed Codesets — Hardware Implications

The X/OPEN NLS specification deliberately makes no attempt to address the device-handling problems that may result from the introduction of new, non-ASCII codesets.

No one wants to have to buy a completely new set of terminal and printer equipment in order to be able to use internationalised software. However, this is exactly what the introduction of NLS implies at those sites where the hardware does not support 8-bit character sets, or where 8-bit codesets are supported but not IS8859/1.

One can expect that it will be a long time (if ever) before the majority of terminals offer IS8859/1 codeset support. In the interim, users will want to make do with what equipment they have. As long as terminals and printers support alternate character sets some means can be found to force (maybe virtual) 8-bit character support.

If 7- or 8-bit devices are to be used, IS8859/1 can be used for internal purposes, and terminal and printer device drivers amended to use lookup tables that enable them to transmit translated character codes to output devices, along with appropriate escape sequences for swapping between codesets, and to decode and translate incoming escape sequences and characters from input devices. This problem has been addressed in the past^[2] and the area is well understood.

6. Limitations of the NLS Specification

The specification for NLS was outlined for the first time in Issue 2 of the XPG in January 1987. Since the standard is very new, it is necessarily somewhat limited in scope. However, future editions of the XPG can be expected to extend the specification into new areas.

Only 22 UNIX commands are specified as having to guarantee processing 8-bit data correctly. It can be expected that this list will be extended in future to cover all the standard UNIX Section 1 commands.

Only one extended 8-bit character set is specified. This set does not provide support for languages that have mainly non-Roman alphabets — Greek, for example, or Hebrew. Further codesets will however be named at a later date.

The introduction of dictionary collation for sorting, etc., rather than machine collation means that the old regular expression syntax for character classes is no longer sufficient. A special syntax will need to be introduced, that uses generic ctype(3C) class identifiers. For example:

[A-Za-z0-9]

will need to be replaced by something like:

[(isalnum)]

The exact form of the syntax for character classes is in the process of being decided by a working committee.

The language announcement mechanism defined in X/OPEN NLS is not very flexible. It is possible to set a new language/culture/codeset combination with `nl_init(3C)`, but not possible to set only a single element of the combination. This means that "mix-and-match" environments are not possible. If a program wants to switch a single element of the local environment, the entire new environment must be loaded into memory. The ANSI X3J311 draft standard for the C programming language

proposes a more sensible solution to this issue (see the next section for more details on the differences between NLS and XJ311).

Problems of coping with mixed character sets and with hardware that does not support IS8859/1 are covered in Sections 4 and 5.

7. NLS and the XJ311 Draft Standard

The ANSI XJ311 Draft Standard for the C Programming Language specifies a number of "international" library routines.

The NLS and the XJ311 internationalisation specifications share the same general architecture. However, they do differ in some minor ways:

- NLS's `nl_init(3C)` function is replaced in XJ311 by the more flexible `setlocale(3C)`, which allows programs to set and reset individual elements of the local customs data.
- XJ311 separates string collation away from string comparison — `strcmp(3C)` and `strncmp(3C)` remain unchanged, while an extra function, `strcoll(3C)`, must be used to collate the strings before comparing them. This is a sensible separation of functionality.
- slightly different routines are used for handling date and time.
- NLS includes some enhanced I/O routines — `nl_printf(3C)`, etc. — that allow parameters to be passed in variable orders to print routines, to support variations in word order between languages. These are extremely useful routines, surprisingly absent from XJ311.

Overall, there are no substantial or irreconcilable differences between the two standard specifications. It is to be expected that they will eventually converge — hopefully taking up the best features from both.

8. 16-Bit Character Sets

At some stage in the future it is inevitable that NLS will have to take the question of the extremely large Far Eastern alphabets into consideration. Since they contain so many characters, these languages can only be represented with 16-bit (or larger) character sets.

16-bit implementations of UNIX already exist, especially in Japan, where a number of companies (including AT&T Pacific) have their own proprietary Kanji solutions. The problem here is suggested by the very word *proprietary*. Many different methods of implementing 16-bit character sets are used and it is difficult at present to predict which, if any, will eventually turn out to be a future standard. Meanwhile, it is a fact of life that there is little or no data compatibility between different vendors' systems, and that software must be rewritten, maybe substantially, to work on different vendors' machines.

Moreover, if UNIX needs a lot of work to make it 8-bit compatible, it needs an order of magnitude more work to make it support 16-bit character sets.

There are certain other problems that may or may not crop up depending on the way in which 16-bit character sets are implemented. These include, amongst others:

- in designing a 16-bit character set, there is a trade-off between efficiency of data storage and efficiency of run-time processing (i.e. either one or the other will deteriorate, in certain circumstances badly);

- byte redefinition may occur. For example, a "/" in the second byte of a character in a 16-bit filename might be taken as a directory delimiter;
- most implementations allow for a mixture of 8- and 16-bit characters, which may cause difficulties in character recognition (and thus especially in string handling), and
- due to the large size of the alphabets, dictionary collation may entail an excessive amount of processing (in fact, there are at least three possible different collation orders for Kanji).

The viability of UNIX in the Far East cannot really be assured until standard codesets are agreed for each country, such that reliable, portable software can be produced, and machines of all persuasions can talk to each other easily.

9. *The Future*

It is the intention of the X/OPEN group to continue to develop and extend the NLS specification. It is likely that NLS and the ANSI X3J311 standards will converge at some future point.

It is to be expected that eventually all the UNIX commands will have to be 8-bit clean. Other areas will be cleaned up to support 8-bit data, most notably and importantly regular expressions and the curses library.

We can also expect to see an internationally accepted set of translations of the UNIX utility error messages, hopefully administered by an international body such as the X/OPEN group. Currently, if any translations exist at all, they are proprietary translations, and so the same error message will appear differently on different systems. However terse and incomprehensible the original UNIX error messages may be, they do have the advantage of being (with some minor exceptions) identical on all UNIX systems — this is a great help to debugging shell scripts, and generally to feeling at home in the UNIX environment. It is obviously vitally important that standards are maintained in this area.

At some stage, NLS will be extended to include other 8-bit codesets, to support such languages as Greek, Turkish, Arabic and Hebrew. The latter two pose new problems, being right-to-left alphabets (e.g. how are left-to-right and right-to-left languages embedded within each other?). Eventually, 16-bit codesets will be introduced.

My personal view is that one day in the future, when storage and processing are *really* cheap, we will see the introduction of a single, literally global, 32-bit character set. This character set would be so large that it could contain every single character of every single alphabet in the world, with some left over for the extra-terrestrials when they arrive. Such a codeset would be capable of referencing other things beyond mere characters — graphical icons, colours, etc.. Gone would be the problems of having somehow to work out what character set someone is trying to communicate with.

"And ASCII shall lie down with Cyrillic..."

10. *Finally...*

Internationalisation has become a real market requirement now that UNIX is starting to become widely accepted and used outside the academic world. There has always been fierce debate as to whether UNIX is a user-unfriendly operating system. If that is true in any way, its solidly American bias must be the most user-unfriendly aspect of all. Despite its limitations, NLS does at least offer a *standard* solution to

the problems of internationalising software.

All the member companies of X/OPEN are committed to supporting NLS on their UNIX systems. According to Mike Lambert of ICL, speaking on behalf of the X/OPEN group at a recent UNIX seminar in London, the members of X/OPEN are committed to conformance with Issue 2 of the XPG by the last quarter of 1987.

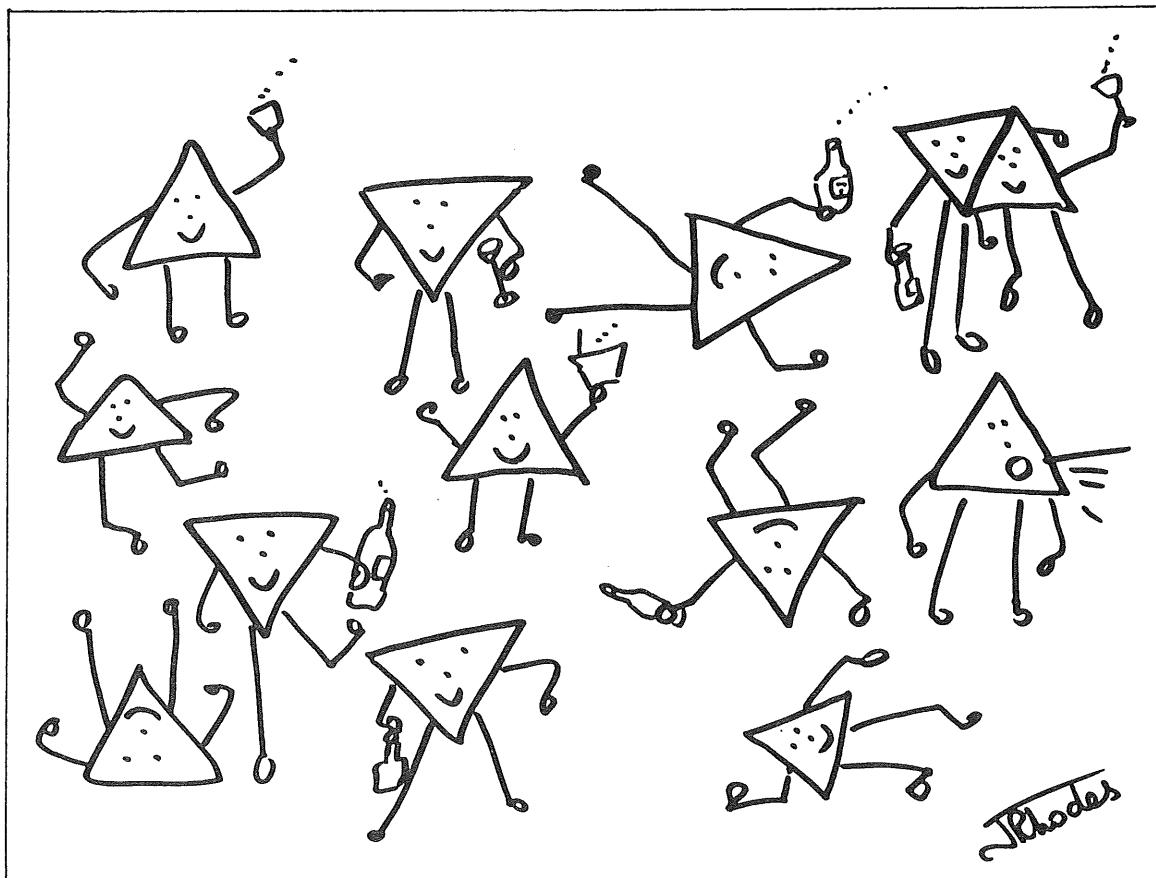
Portability of software that utilises the NLS interface will thus be guaranteed across the entire range of X/OPEN companies' UNIX systems. In addition, it can be expected that other computer manufacturers will follow the X/OPEN lead and provide NLS on their systems.

X/OPEN also guarantees a future upgrade path with backwards compatibility. This, in conjunction with the portability of NLS applications, ensures protection of software investment.

Although the present NLS specification is not perfect and is limited in many ways, it represents at least a pragmatic solution to the problem of internationally acceptable UNIX.

References

- [1] *XVS Supplementary Definitions* X/OPEN Portability Guide, Volume 3, January 1987, Elsevier Science Publishers B.V.
- [2] Conor Sexton *European Languages in UNIX* Proceedings, EUUG Autumn 1985 Conference, pp. 195-210.

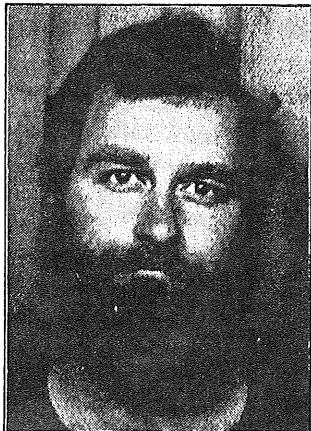


UNIX Grows Up

Grouse: Messages and Prompts in Programs

Alain D. D. Williams

...!mcvax!lukcl!nset!phcompl!addw



Parliament Hill Computers

Alain Williams is an independent consultant specialising in UNIX and C. His interests are: finding what people's problems are, and building good tools to solve them once and for all; doing the job properly as it takes less time; drinking cider; and jumping out of small boats into a cold sea.

He is the editor of the EUUG newsletter.

Editor's Note:

This is a reprint of a paper which was given at the UKUUG meeting, 15 December 1986.

Grouse can be used in programs to replace printf() statements, the text for the messages coming from files. Printf() style argument substitution is available, arguments can appear in any order.

This paper discusses:

- *What grouse is like to use.*
- *Implementation.*
- *The advantages of choosing compiled text files.*
- *The way in which grouse is used by higher level functions to permit automatic interpretation of errno in error messages.*
- *A greatly simplified prompting procedure allowing help to be taken with little action by the applications programmer.*
- *Language independent option prompting trivially supplied.*
- *The way this is tied into a help package.*
- *How grouse can easily crash a program and why this is, in practice, not a problem.*
- *Portability of code and message files.*

*Version 1.4
Updated 12/9/86*

I like work: it fascinates me. I can sit and look at it for hours.
Jerome K. Jerome

Grouse: (noun) grumble (slang) (Oxford English Dictionary).

1. Introduction

What follows is basically an exercise in laziness, one in which I was prepared to invest a large amount of effort. There are two major topics in this paper: *grouse* a text from files subroutine, and a set of subroutines built up on top of *grouse*.

1.1 Ancient History

The initial impetus came from a set of routines which I first wrote in 1979. These were to help me debug BCPL programs. I put assert type traps in them which printed a rude message, and quit. Often needing different amounts of information I soon got bored with recompiling, something which took a long time — even with an illegally raised priority.

Editing text files was fast. So I put all the messages into a file (along with flags requesting things like dumps, and stack traces) and wrote a routine to find and print numbered messages. Because this was always used in a situation where programs complained about errors I called the routine *grouse*.

The next ingredient is numbering arguments. Thus referenced I could substitute and format them using `writef1`. This is easy to do in BCPL, which implements a word based machine architecture: I just indexed on the address of the first argument.

I then became a UNIX® acolyte and implemented *grouse* on a PDP11. The way the C compiler worked allowed the old indexing trick, but different objects had different lengths. This caused problems. I wrote a compiler to fudge the issue, it massaged the numbers. A compiled file also offered greater efficiency of message location and meant that the run time code could do away with much format checking — code size ever a problem on a 64K machine.

1.2 The Present Day

As a simple message system *grouse* is nothing unique. Its great usefulness has come from a set of subroutines which all use *grouse* as a base. The most interesting are those concerned with prompts in interactive programs. The programmer is presented with a simple, clean, standard interface; he need not care if the actual form of the prompt is changed to suit the flavour of the month.

Grouse and its family are in the throes of a third birth. This opportunity has allowed me to try and avoid past problems (and introduce new bugs). It is this version that is described here.

2. Overview

This section should give you an impression of what using *grouse* is like, and the effort involved in using it.

1. The BCPL equivalent of `printf`.

```

/* Report the open failure */
egrouse(stderr, GR_OPEN, (char*)errno, progname, file_name);

/* What does he want to do ? */
switch(gaskq(GR_PROMPT, (char*)0, name, old, new)) {
case 1:      /* He agrees with me ! */

.
.

break;
}

```

Figure 3. Grouse

Notes:

- There is less code to type.
- The message text is not obvious.
- The reply has been checked, there is no need for an error condition.
- Both prompt and reply are language independent.
- Help comes free, the programmer does not need to do anything to get it.
- The second argument is always `errno`, even if it is not wanted. It must be cast as shown.

Unfortunately the immediacy of having text in the code is lost. You have to look in a message file to inspect or create that text. The messages are referenced symbolically by `#defines` — which are probably contained in a third file. This requires a disciplined approach and a little documentation.

Grouse can also be used to obtain constant text which may be expected to change from one situation to another; e.g. the names of days of the week, customer name, file/directory names.

3. Message Files and Message access

A message file exists in two forms: source and compiled.

There are two compiled files that grouse may use at any one time. The idea is that one contains text that may be common between a group of related programs. Grouse decides from which file to obtain the message on the basis of the message number: numbers in the range 0 to 19999 are read from the first file, numbers 20000 and up from the second.

The way that I have used this feature is to put in the first file text used in many places, e.g.: strings representing the values of `errno`, prompts for help routines (being library routines common to a suite of programs), a language collating sequence.

There may be large gaps in the message numbering sequence. This is very important in providing a stable environment to existing code, yet allowing new messages to be added to a common part. In previous versions a break in the sequence meant a corresponding waste of disk space.

2.1 The Bad Old Days

This is the era in which most programmers still live. Two examples follow: they are fragments of code typical of what I have seen many people generate — though they usually don't have as many comments.

```
fprintf(stderr, "%s: Cannot open '%s' as %s\n", progname, file_name,
       sys_errlist[errno]);
```

Figure 1. An error reported

```
name = "master";
for(;;) {
    position(22, 0);
    line_clear();
    position(23, 0);
    line_clear();
    position(22, 0);
    printf("Oh %s: shall I rename %s as %s ?\nReply (Y)es or (N)o : ",
           name, old, new);
    switch(getchar()) {
    case 'y':      /* He agrees with me ! */
    case 'Y':
        .
        .
        break;
    default:        /* Idiot can't read the options */
        position(21, 0);
        line_clear();
        position(21, 0);
        printf("Please reply with one of the options below");
        name = "idiot";
        continue;
    }
    break;
}
```

Figure 2. A Question Asked

Notes on the examples:

- The message text is embedded in the program.
- In Example 2, the programmer needs to cope with the error case himself. This results in extra, unwanted, code. A `for` loop is needed with resultant extra indenting, control structures, and so complexity.
- An explicit test needs to be made for both upper and lower case.
- There needs to be screen positioning code, etc.

2.2 Modern Times

With `grouse` the two examples may be coded like this:

3.1 How to Find a Compiled Message File

This is a big key to (human) language independence. The name of the file opened is built up of: an application supplied *grouse* file name; a constant part; a suffix, and two environment variables. These variables are *LANGUAGE* and *MSG_LEVEL*. Standard uses are: the first names the user's mother tongue, and the second is a measure of his incompetence.

In this way an individual's environment can be set so that he uses the machine (reading, and replying) in his native language. Similarly it is trivial to arrange for a beginner to see prompts that would be unacceptably long to an expert. So two people running the same application, on the same computer, at the same time could see something very different.

3.2 The Source Message File

This is a plain text file created with a suitable editor. The messages are delimited on a line basis. They contain the text that is to be retrieved by *grouse*; this may be several lines long. The backslash \ character is used in a similar, but extended, way to the C language.

As with *printf*, arguments are introduced with a %. However, the character immediately following indicates the position of the argument as passed to the top level function. If any arguments are unused, what they are must be indicated in a special way.

There are several different message formats. This corresponds to the different higher level function that will end up using it. The type will be checked by the message compiler.

What does it look like?

```
%1s: Cannot open '%2s' as %0s
```

Figure 4. Grouse Text for the First Example

```
Oh master: shall I rename %1s as %2s ?
Reply (Y)es (N)o (H)elp : ^ l|n|h?rename_help
```

Figure 5. Grouse Text for the Second Example

Notes:

- The arguments are indexed by the character after the %.
- The arguments can occur in any order.
- You don't have to use the first argument.
- Message length, in lines, is not important to the application programmer.
- In choice prompts the reply is encoded in text after the prompt.
- Help is obtainable by entering H. The file containing the help text is also encoded as part of the reply text.

3.3 The Compiled Message File

This is what the *grouse* subroutines see. The main advantages of a compiled message file remain as they were originally:

- The ability to work on a non-word-based architecture. The message numbers are changed to make the old, simple indexing scheme work. With some awkward

machines it is necessary to also include extra information on length and positioning.

- The message file contains information that allows `grouse` to calculate the file offset of the desired message. This is much more efficient than sequentially searching for it.
- The compiler has syntax-checked the messages, and made them easier to parse for use.

The compiler offers facilities such as argument type cross check (with the corresponding message in another file) and symbolic referencing of one message to another.

There is also a certain amount of control information, including version numbers and the compilation date.

4. Higher Level Functions

This is where the fun starts.

4.1 Egrouse — Errno Interpretation

You have already seen this in the new coding method for the first example. `Egrouse` is functionally the same as `grouse` except that it takes its second argument to be a value of `errno`. It calls a function to return a string into a secondary buffer, which provides an interpretation of the number. It then overwrites that argument with the address of the secondary buffer and calls `grouse`.

It is because of this overwriting that, as you will have noticed, `errno` must be cast to `char*`. Many of the high level functions use `egrouse` and so need a cast value of `errno` passed to them. This is initially confusing, but soon becomes natural.

If the situation has no use for a value of `errno` one must still be provided. In this case zero may be passed. The empty string will then be assumed, i.e. no file access will occur.

4.1.1 Errno Extended

Not being restricted to the list of messages that I found in `sys_errlist` came as a boon. I was able to easily add my own definitions of values for `errno`.

Up until now if something went wrong in a library routine it had to report the problem itself. This might be difficult if it was something of general use — what method should it use, should it complain at all?

All that was needed was for the routine to set something meaningful into `errno` and return failure, this being transmitted up a calling sequence. At some convenient level, and if appropriate, the problem could be reported with `egrouse`. It made no real difference whether the error status was one of my own invention, or one from the vanilla list.

4.2 Gaskq — Prompts

A common event in an interactive program is prompting of the user for information. This can occur in several different ways, the most common of which is a prompt for a choice of action. This is what was happening in the second example. `Gaskq` `egrouses` the message into a buffer and calls `askq`. The latter puts the message on the terminal and looks at the reply.

A list of acceptable replies is encoded at the end of the message. If a match is found the default action is to return the position in the list of the matching reply.

Thus, in the example, `y` returns one and `n` returns two. This action may be changed by following the reply by an action string. The one illustrated directs `askq` to call the help function, and gives the name of the help file to use.

Other actions include: reprompt with another prompt and return a number other than that deduced from the positional rule. It is possible to specify a default.

Related functions prompt for a string or simply display a message.

4.3 Help

One of the `askq` built-in actions is the provision of help. This is obtained through a standard interface from `gaskq`. The default help routine allows the user to peruse or dip into the help text as he wishes, moving back and forth in the text, following links into other files for explanation of common parts.

The routines prompt the user to direct his perusal using `gaskq`. Help on how to use help is trivially provided.

A "mini help", i.e. a slightly longer, more explicit, prompt can be obtained by switching to another message number to use as prompt. This will be displayed, again using the same arguments as the original.

The calling program knows nothing of all of this as it is done by `askq` directed from the message file.

4.4 List Building

There are occasionally lists of words that a program may need to get going, for instance the names of the days of the week, or column headings. These can be obtained from `grouse`.

The action in each case is similar, i.e. read the text, allocate storage, and split it up into different elements of a string array. A routine is available to do this, and it checks more than one might normally do on a once off basis: are the number of elements between certain limits, is any element too long?

5. Programming Considerations

As you have seen `grouse` may be used in a similar way to `fprintf()`. To be useful in the higher level functions, the text needs to be put into a character buffer — as with `sprintf()`. The trick employed is that if `grouse` sees that the message number is negative it considers its first argument to be a `char*` rather than a `FILE*`.

The programmer has to specify the names of the two `grouse` files. This is managed by setting a name into a global structure. There is no need to explicitly open the files.

The defined arguments to the open routine are a name and a suffix. This routine is thus available for finding other sorts of file in a similar way, for example, help files. If screen layout files are used, this area of the MMI can also vary in the same way as the messages and prompts.

5.1 Ranargs

This is a facility which enables the arguments to a function to be accessed in a random order. It is designed to have a feel similar to `varargs`.

Define an argument list as being the arguments that a function has, possibly omitting a specified number at the start. There are two things that `ranargs` lets you do with a list of arguments: access a specific argument, and pass the list to

another function. The latter ability is vital in constructing the higher level functions. This is all simple enough on machines where the arguments are placed, one after another, on the stack, and there is no greater alignment needed than that of an integer. If these conditions are not met, the `ranargs` macros become complicated, possibly invoking functions. To get some idea of the problems involved consider why, on a machine where doubles indeed have a greater alignment than an integer, the compiler may have to leave a gap on the stack before it.

The lack of exact control over what the compiler does is one of the consequences of using a high level language. The problems I have had are similar to those who try to do single precision floating point in C, and I know of several device drivers that cannot be optimised.

5.2 Errors

As with `printf()` `grouse` returns -1 if it can't do what it was asked to. Frequently programmers don't bother to test return values because "It always works", and because it is a chore. The problem becomes even more difficult if an error is detected by a higher level function; what should it do?

Firstly, the external result is always sensible. Even if the function's return value is ignored there should be no bad effects, e.g. `grouse` never leaves an unterminated string. Next, on finding trouble all the routines in the package call an error function, suitably indicating the problem. The library default of this function does nothing, but it can easily be replaced by the application programmer.

5.3 Robustness

The code in `grouse`, and the higher level functions, has proved solid. The problems that are likely to arise are in their use; probably not in the coding itself but through the message file — particularly if this gets changed. The three areas with greatest potential for `grouse` crashing a program are:

1. Inappropriate use of an argument.

For example trying to output an integer as a string. In practice this is rare as the programmer will normally create the message file when he writes the code. If the messages are changed by someone else, the compiler cross-check action can be used to pick up any mistakes.

2. Overwriting the end of a string buffer.

If text is going to a character buffer `grouse` does not check that it does not overflow the buffer. In four years of having a primary buffer of 550 bytes, and a secondary one of 150, I have not experienced any problem because of this. The choice of buffer size is important and needs to be carefully considered; those changing the message file should be aware of the problem.

3. A message not being found.

This could be because the message file is not accessible, or a message is missing. This occasionally happens (often in `gaskq`); the usual result is that at that point the program always takes a certain branch — occasionally leading to an infinite loop. This situation is now easier to trap with the introduction of the `grouse` error function.

If the file exists, the usual cause is running a new program with an old file. The version number feature is designed to avoid this.

5.4 *Malleability*

Much recent effort has been made in the area of ease of programmer customisation: how he can best make it do what he wants. This has been done without making simple or standard use any more difficult. The programmer has much control, but only if he wants it.

Some standard routines can easily be replaced by the programmer. One version of the grouse open function that I have searches down a path taken from the environment in a similar way to the shell when finding commands.

`Askq` (and hence `gaskq`) is independent of the method of communication with the terminal. Thus by replacing the low level display routine it is easy to change the user's view of the program from dumb terminal, to a cursor addressed prompt line scheme, to a mouse-driven pop-down menu approach; all involving no change to the main body of the program. It may be expected that the message file may have to change to match the different styles, though I am working on a way of transparently incorporating alternatives into the message file.

The grouse package is designed to cope with 16 bit characters. There is no hard-wired dependence on special characters or character size; these may exist in utility functions called by the package. There is still much work to do in this area: I have still to find a 16 bit terminal on which to test this.

5.5 *Portability*

There are two aspects here: the code and the message files.

With the invention of `xanargs` the code has proved to be portable. Other than this there is little in the code that is contentious — except to `lint` which is vastly unhappy about some of the strange type casts.

The message files are only portable in source form. This is because of the massaging needed on message number.

6. *How does it Work in Practice?*

A version of grouse has been available for four years. During this time it has been used on two major projects, both involving several large programs, and a total of some 1500 messages. The latest version is too young to have been used in anger, but was designed to alleviate problems or restrictions in earlier attempts.

6.1 *Programming*

The main problem has been one of initial learning and acceptance. People get set in their ways, and I found that, even with some encouragement, some are reluctant to use new tools. There is a short learning phase (`grouse` has always been well documented), and a certain discipline is required to ensure that the symbolic numbering does not get out of step with what is in the file. Once this has been done the attitude changes to "this is the easier way".

If a model of interaction is chosen that is not available as standard, then some new low level display routines need to be written. This can be a barrier to those with the "I'm only interested in today's problem" attitude — especially if they know that someone else is to maintain and possibly modify the code for another language.

6.2 *Speed*

I have not done much quantitative analysis. However, is it pleasing to report that an interactive `groused` program does not appear to be significantly slower than a hard-wired equivalent. This is even in a situation where a composite message may

be produced from several fragments.

6.3 *Altering Messages*

Documentation is the key. If this is not complete and explicit then references need to be made to the code: having a quick peep is error prone — even for an experienced programmer. This applies if the text is being translated or the style is being changed. Rewriting a *grouse* file is not a quick and easy task. Message files tend to be large, and the translator needs some understanding of `printf`. I have found that, with a little tuition, even managers can do this.

7. *Conclusions*

- It has been worth while. The increase in productivity through using the tools discussed here is great. More work is needed to take them further, both in the programmers' tool kit and to simplify the task of the application customiser.
- C has been criticised for being too low level a language for applications programming — I quite agree. However lack of precision can cause problems, as evidenced by `ranargs`. What to do is not obvious. C has served us well and doubtless will for some time yet.
- There are many things which I find boring: writing similar code several times is one of them. Subroutines were discovered years ago. I am still surprised by how often people who ought to know better will still "lift and hack" code rather than "doing it once and doing it well".

Another Proposal for a News Scheme

*John Collins
jmc@xisl.co.uk*

Xi Software Limited

1. The Problem

There are at present two options regarding the news:

1. Don't get it at all.
2. Get all of it.

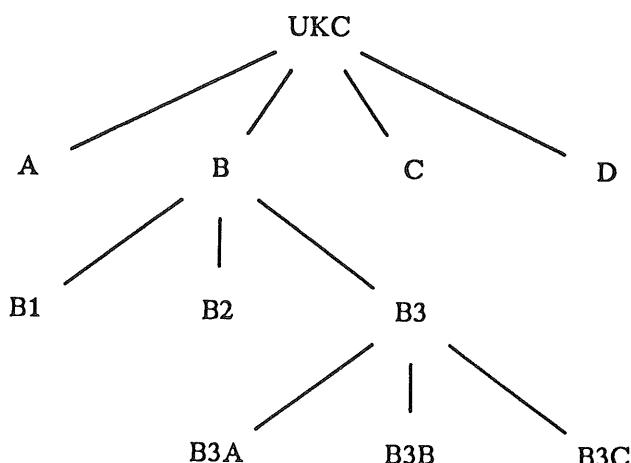
Assuming that the reader wishes to avoid option 1, option 2 may be undesirable for any or all of the following reasons.

1. There is so much of it he/she doesn't have the disk space.
2. It takes so long to transmit that the phone is constantly engaged.
3. The phone bills involved are prohibitive.
4. Having got it, about 5% is interesting and the remainder is about someone selling a second-hand goldfish bowl on the other side of the world (usually an American with an imperfect understanding of the relationship "usa != world").
5. People at various sites spend hours every morning wading through all these news items trying to find interesting ones instead of doing useful work.

The following is my suggestion for a way to compromise between the two extreme options with regard to the news.

2. The Proposed Solution

For the sake of argument, let us suppose that a section of the network is as follows:



At present, sites A to D obtain news from UKC, and site B sends a copy to each of sites B1 to B3, site B3 sends further copies to B3A to B3C. This represents a great

deal of duplication — especially if only a few items are of interest at each site.

The proposal is that site B continues to receive the news from UKC as at present (note: no work for UKC!), but instead of passing it on, passes a "headline file" to each of the sites on the line. This file is a text file, derived from the newsgroup name, the article id and the subject line, possibly of the form:

mod.sources

xyzabc.1234 Public Domain ADA compiler Part 01/87
xyzabc.1235 Public Domain ADA compiler Part 02/87

.....

misc.rumors

foobar.1234 AT&T to give free source licence with every 3B
blech.1383 IBM to pull out of computers
frozb.3934 Re: IBM to pull out of computers

.....

Users on each site then put up this file and request any articles of interest, using a full screen program. The requests are collected at each site one down from the "headline sender" site. Thus each of sites B1 to B3 collects requests relating to the "headlines" sent by site B.

At the sites B1 to B3 the requests are "weighted" using some or all of the following factors:

1. Number of requests for each item.
2. Age of item (so that old news decays in importance)
3. Cost of transmission to requesting site.
4. News group importance.
5. Item size in bytes.

For each site B1 to B3 there is a "request threshold"; if the total weight of requests exceeds this, then the item is obtained from site B.

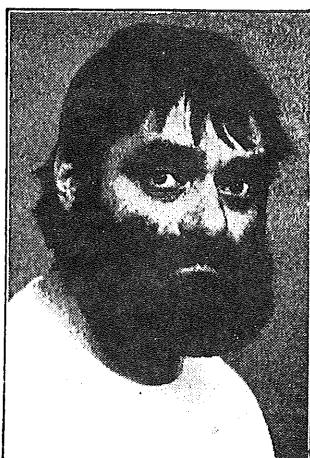
Likewise for each site B3A to B3C there is maintained at site B3 a threshold for passing the item on.

This can all be extended downwards, say from one of sites B3A to B3C, and in that way, subject only to a day or so's delay in getting the news, some control is obtained in the volume of the news.

EUUG

*Teus Hagen
EUUG Chair
teus@oce-rd1.nl*

EUUG executive board



Teus Hagen was born in 1945. He graduated from the University of Amsterdam. His computer science knowledge was gained mostly at the Free University of Amsterdam.

He has been involved in UNIX since 1975, at the Mathematical Center (currently CWI) in Amsterdam (the second UNIX site in Europe?).

Since 1985, he has worked in UNIX companies, and is currently at Oce Holland (Office Automation).

Teus has been involved with the EUUG since 1977, and has been the EUUG chairman since 1985. He was responsible for starting Eunet at Paris in 1982, and arranging the current structure of the EUUG in 1983.

1. Academic, that is a long time ago

Some old hackers say the group started in Edinburgh, others say it was in Canterbury, continental hackers believe it was in Amsterdam, but nevertheless all are right. It was about ten years ago that some enterprising people started to complain about UNIX. They exchanged software on huge disk cartridges in order to adjust their feelings for the next time. From about 15 academic students and professors from a few countries in Europe, the EUUG went in ten years to 2500 members (mainly from institutions and industry) from almost every country in Europe. In *Dublin* the EUUG will celebrate its tenth anniversary. That event should not be missed!

Apart from the countries behind the iron curtain, and Portugal, a national UNIX group can be found in every country, all bundling their common efforts under the umbrella of the EUUG. Each national group has appointed her representative to the *governing board*. The daily bother is left to eight persons in the *executive board*. They meet about every two months in order to keep the EUUG going. The secretarial assistance comes from the people at Owles Hall in England. Those people really make a good job of all the peculiarities arising from all those languages and national specialities.

With the big growth and the success of the organisations and the EUUG one can expect some growing pains. On the national level, there is plenty work in combating the national problems. This means less effort to be expended on the European level. Also the enormous amount of work which accompanies the EUUG work is very hard to see. Conferences need a lead time of at least one year. To get plans adjusted and agreed takes a very long time on the European level. The effect of all that work is seen too late. Still, the enormous encouragement from the national level is a big motivation for the executive board to continue.

COUNTRY	GROUP	MEMBERS	COUNTRY	GROUP	MEMBERS
Denmark	DKUUG	164	Britain	UKUUG	304
Finland	FUUG	131	Ireland	IUUG	43
Sweden	EUUG-S	195	Swiss	UNIGS	28
France	AFUU	362	Germany	GUUG	392
Italy	I2U	291	Austria	AUUG	20
Holland	NLUUG	190	Belgium	BUUG	31
Iceland	ICEUUG	10	Norway	NUUG	68

EUUG Membership January 1987

Normally the time spent by board members on EUUG tasks is too long (usually about 20% of their working hours). One can expect that more full time paid personnel will be needed in the future to limit their EUUG working hours.

Clearly the conferences, with their technical contents, industry presentations, and tutorials, are the first things from the EUUG one sees. The EUUG newsletter is felt not to be distributed frequently enough. More effort is being put into meeting that problem. More EUUG publications similar to UNIGRAM weekly, a glossy journal (probably UNIX Review), a UNIX technical journal, a catalogue and a European UNIX diary, can be expected.

Unexpectedly, the EUUG software distributions are meeting the needs of the members. There are thoughts of making that software available on other media as well as magnetic tapes. Also there are discussions going on about making them available on EUnet.

EUnet is being registered as a trademark throughout Europe. The network is recognised as one of the largest in Europe. Gateways exist to other networks such as CSNET and DFN. For every country a domain is being registered (was Holland with the domain *nl* the first?). The backbones are very well interconnected. Sometimes I'm a bit angry that my electronic mail takes more than an hour to get answered by someone in France. Sometimes I get worried that the people of a particular company in the US have gone home before they have had a chance to read my mail. Due to the efforts of the people involved directly with EUnet, email has less failures than one could imagine some years ago. EUnet is very successful.

Sometimes it is necessary not to put some of the work in the direct sunlight. That certainly is true for the regular meetings with the organisations in the front lines of the UNIX fields: for instance UNIX Europe, AT&T International, but also X/OPEN. Especially, perhaps, X/OPEN needs some more input from the user group. More manpower is needed to support that effort. Think about the enormous experience Europe has with international peculiarities (character sets, languages, etc.).

The EUUG has recovered from her past financial problems. The financial situation is sound now, so we can direct our attention more now on those problems and tasks which are common throughout Europe. For just that reason, there is an EUUG. The enormous stimulation, the contributions, from the national groups make this UUG organisation a success, which should not be missed in any country.

Progress of ANSI/ISO C Standardisation

Cornelia Boldyreff
...!mcvaxlukcl!reading!uoseev!corn

Department of Electronic and Electrical Engineering
University of Surrey
Guildford GU2 5XH

1. Brief Historical Background

There exists a high degree of homogeneity between various implementations of C for a variety of reasons:

- the common origins of C compilers;
- its link with the UNIX system (it has been remarked that successfully compiling the UNIX system is quite a rigorous test for a C compiler);
- C is used as a vehicle to achieve portability in general which militates against adding non-standard extensions;
- Dennis Ritchie and Brian Kernighan's clear exposition of the C language.

The latter's book, "The C Programming Language", has become an informal standard for the language; it is not uncommon for suppliers of non-UNIX C compilers to assert that they support Kernighan and Ritchie. This book was published in 1978 and as a standard is becoming somewhat outdated.

It formed the base document for the ANSI C standard committee's development of a standard for C. They also drew on work by the US commercial UNIX users group, /usr/group; particularly for the definition of the C library. The ANSI effort has received support from AT&T as well as major C compiler developers, suppliers and users including UK companies: ICL, The Instruction Set and Edinburgh Portable Compilers. In December 1985, ANSI proposed a New Work Item on C to the International Standards Organisation based on their work.

2. Introduction to Work of the BSI C Panel

The BSI Technical Committee on Programming Languages, IST/5, had been monitoring the progress of C standardisation efforts prior to the proposal by ANSI of an ISO New Work Item on the programming language C. Following approval of the C NWI in April 1986, an ISO Working Group on C was formed; and the formation of the BSI C Panel was set in motion. The role of the C Panel is to provide a UK focus for contributing to the progress of an ISO Standard for C. This panel met for the first time in the summer of 1986. The first meeting of the ISO Working Group was in September 1986; countries represented were the USA, Canada and the UK. The ISO standard work is progressing in parallel with that of the ANSI X3J11 Committee. It is very much a collaborative effort as the draft proposed ANSI C standard is the basis for the ISO standard rather in the way that the BSI Pascal standard was the basis of ISO Pascal.

The BSI C Panel

- is a representative group of UK experts including commercial, industrial and academic users of C as well as suppliers and developers of C compilers;

- meets informally and has no official BSI status; all members of the panel act in a voluntary capacity usually supported by their employers;
- reports regularly to the BSI's Technical Committee on Programming Languages, IST/5, through its Convenor who is a member of IST/5 on its activities and the progress of the C standard;
- advises IST/5 on issues concerning C and ISO ballots relating to C;
- monitors progress of and contributes to the ANSI work on C;
- contributes to the progress of the ISO C Standard through participation in the ISO Working Group on C as individual experts with "awareness of UK reactions".
- collaborates with other UK BSI committees concerned with C related standards work; for example, graphics standards with C Bindings, and the proposed POSIX standard.

The C Panel meets quarterly preceding ANSI (ISO) meetings and BSI IST/5 meetings. Panel meetings are usually attended by a dozen or so members — the panel officially has 16 members. New members are always welcome. Membership is considered to lapse if a member does not attend for three consecutive meetings of the panel. (Interested parties could contact the author who is convenor and chairman of the C Panel.)

3. C Standard Open Meeting

The British Standards Institution's C Panel organised a one-day "Open Meeting" on the proposed C standard to coincide with the BSI's publication of a Draft for Public Comment on the programming language C. The meeting was held on the 9th February 1987 at City University, London.

The keynote speaker at the meeting was Dr P. J. Plauger, President of Whitesmiths Ltd. Bill Plauger is a prominent member of the ANSI committee, X3J11, which drafted the proposed C standard, acting as secretary to X3J11; and chairman of the C library sub-committee. In his opening lecture, he gave delegates an overview of the current draft C standard concentrating on major decisions reached by X3J11 and issues which had taken up the most "air time" in committee meetings over the past three years. He enumerated the tenets of the philosophy which has guided X3J11 in their efforts to standardise C as follows:

- Codify existing practice.
- Existing code is important.
- Portability needs a "fighting chance".
- Non-portable code is OK, too.
- Quiet changes (to C) are bad.
- The standard is a treaty between implementor and programmer.
- The "Spirit of C" is important.

He discussed three major decisions made by X3J11 regarding characteristics of the C machine, C programs' conformance to the standard, and implementations of C. A "C machine" has 8 bit or larger bytes; "no holes" in objects (i.e. except for bit fields, objects are contiguous sequences of bytes); performs weighted binary arithmetic; and has an arbitrary character set. A C program is either strictly conforming to the standard i.e. fully portable; conforming; undefined, or erroneous. An implementation of C may be either hosted or freestanding.

The standard has endeavoured not to radically change the C language; major issues which have concerned the committee identified by Plauger were:

- Conformance issues;
- Widening rules;
- Preprocessor issues;
- Library issues.

In conclusion, Plauger explained the rationale behind the introduction of function prototypes to the C language by elaborating the committee's own version of the US Supreme Court's Miranda Ruling. He also listed extensions considered by the committee which failed to gain approval showing that there was scope for framers of the C standards to come in the 1990s.

Cornelia Boldyreff, the Convenor and Chairman of the BSI C Panel, spoke briefly introducing the work of the BSI language panel concerned with C standardisation. The BSI C Panel was formed in the summer of 1986. Its role is largely advisory: it advises the BSI's Technical Committee on Programming Languages on issues concerning C and ISO ballots relating to C. It monitors progress of and contributes to the ANSI work on C; and panel members contribute to the progress of the ISO C Standard through participation in the ISO Working Group on C as individual experts with "awareness of UK reactions".

The morning session was concluded by John Souter of the BSI's Certification and Assessment Service addressing the issue of testing conformance to standards by language processors. He outlined the work of the BSI evaluating potential candidates for a C Validation Test Suite. According to Souter, the USA validation service is planning to follow the British lead in establishing a test service for C language processors.

The afternoon session was given over to discussing the three main aspects of the standard, dealing with the C language, the C library, and the C preprocessor. Mike Banahan of The Instruction Set addressed the C language and the C preprocessor in two lectures; and Bill Plauger spoke again in greater detail on the C library.

Banahan reiterated that it was not the intention of the committee to radically change the C language; he reassured the meeting that much of Ritchie's original description of C could still be found in the text of the draft standard. His lecture concentrated on illustrating key points where the language has changed.

In his introductory remarks on the work of the library sub-committee, Plauger mused given his involvement in the development of the Whitesmith's C library, some must have seen his selection as chairman of this group as comparable to putting "a fox in charge of the hen house". Of particular interest to the international C user community were the ways outlined by Plauger in which the committee had addressed the issue of "Internationalisation" by inclusion in the library of a runtime selectable locale. The library defined in the draft standard has had ASCII dependencies removed; is more complete; and covers domain and range errors in mathematical functions. Areas of the library identified by Plauger as still needing attention included functions to restore calling environment: `setjmp/longjmp`; signal handling: `signal/raise`; and variable arguments handling macros.

Describing his reactions to the preprocessor defined in the draft, Banahan speculated that here the ANSI committee had used great artistic licence. Existing preprocessor code would be broken. On the positive side, Banahan suggested that now the preprocessor was better described. By a series of interesting examples, he illustrated

features of the proposed preprocessor.

The final lecture in the afternoon session was given by David Tilbrook, a veteran C programmer and UNIX guru. Tilbrook gave a historical perspective to the development of C from its early PDP-11 days to the present. Tilbrook made the point that most early C programmers were experienced professionals while today C is being used by programmers without knowledge of any other language and little understanding of the underlying machine on which their programs will run. These programmers will certainly benefit from the proposed standard.

The open meeting concluded with a panel session including all the speakers. The audience raised a variety of questions ranging from when will AT&T supply UNIX with a standard conforming C compiler to the relationship between the proposed standard C and Stroustrup's C++. The BSI organisers co-ordinated by Paul Neale received a vote of appreciation from the chair for their efforts in contributing to the success of the meeting.

A major objective of the C Panel in organising this open meeting was to promote standardisation of C and facilitate UK public comment on the draft standard. The meeting was attended by the C user community at large in commerce, industry and education as well as C compiler developers and suppliers in the UK.

Interested BSI members and members of the public can obtain copies of the current C standard draft directly from the BSI. Public comment is invited; and all comments received by the BSI will be processed by the BSI C Panel and copied to the ANSI X3J11 committee.

4. Future Meetings in 1987

ANSI X3J11	ISO Working Group 14	BSI IST/5/14 "C Panel"
		5 May 87
	← June 87 → Joint ISO/ANSI Meeting Paris	
14-18 Sept 87 Boston		11 Aug 87
7-11 Dec 87 Phoenix		3 Nov 87

Once an ANSI standard for C has been approved, it is likely to be put forward for registration as a Draft International Standard for C and, following review and approval, become the basis for ISO C. The ISO Standard for C would then be adopted as a BSI C standard. It would be subject to regular standard review procedures; and as long as C continues to be a "living language" subject to new developments, the work of the BSI C Panel will continue.

5. Summary of Progress to Date and Future Timescales

The standardisation process is essentially an iterative one; the essence being to achieve agreement between all parties — in a word: consensus. The figure below charts progress to date and milestones for the future.

ANSI	ISO	BSI
C project approved and X3J11 committee formed (1983)		Monitoring C Standard Progress
C Language Information Bulletin published for informal comment (July 1985)	NWI on C proposed by ANSI (Dec 85) NWI approved and Working Group 14 formed (Apr 86)	
X3J11 reach consensus on Draft Proposed Standard (Sept 1986)		BSI C Panel formed (July 86)
Public Review of dpANS (7.11.86-7.3.87)	dpANS submitted as Working Paper Registration of dpANS as ISO DP Letter ballot 11.86	BSI publication of dpANS as BSI Draft for Public Comment (Jan 87) C Panel Open Meeting (Feb 87)
ANSI Standard (end of 87?)	Draft International Standard (end of 87?) International Standard for C (sometime in 88?) → BSI C Standard	

The iteration involved in the process consists of several loops. Within the ANSI work, there is a tight inner loop where agreement on the proposed standard must be achieved within X3J11 and an outer loop where public approval is sought. Internationally, agreement on the standard must be achieved within the C Working Group and the standard must gain approval from the member countries of ISO.

X/OPEN — What, Who, Why, When

John Tottenham

ICL

1. *What is X/OPEN?*

The X/OPEN Group is a unique consortium of eleven of the world's major information systems suppliers who have come together to agree on standards for operating systems and applications portability.

X/OPEN is not a standards setting organisation, it is a joint initiative by members of the business community to integrate evolving standards into a common, beneficial and continuing strategy. The keystone of this strategy is the common Applications Environment, a complete environment for the easy development, porting and running of applications across systems from all X/OPEN Group members.

2. *Who are the members of X/OPEN?*

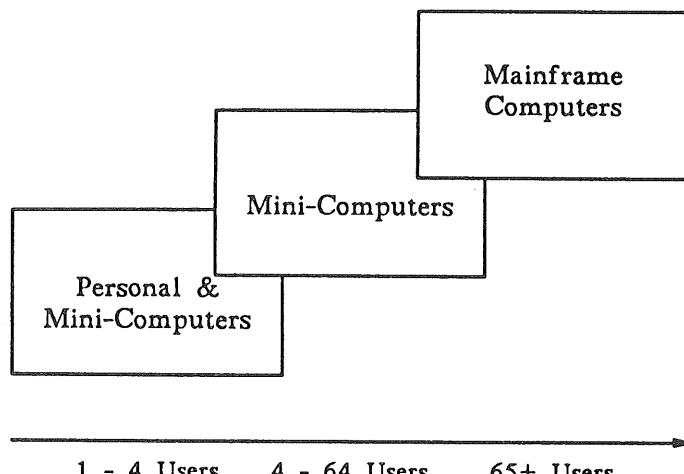
Currently, the eleven full members of X/OPEN are: AT&T, Bull, DEC, Ericsson, Hewlett-Packard, ICL, Nixdorf, Olivetti, Philips, Siemens, and Unisys. All these eleven members have made substantial financial and technical commitments and will continue to do so, providing users with a higher level of insurance for the future supply of systems based on industry standards than a single manufacturer alone could do.

In addition to the full members, numerous other companies and consultants in the Information Technology Industry have contributed to the technical and marketing programmes of X/OPEN.

3. *Why was X/OPEN formed?*

The formation of the X/OPEN Group was a direct result of two major changes in the Information Technology Industry in the early 1980's, the emergence of the Department as a large scale user of computer systems, and the growing market fragmentation caused by propriety operating systems, particularly amongst the small to medium sized mini-computers.

The three major categories of systems are simply identified according to the number of terminals attached:



This breakdown is significant since it maps the current areas where market dominant or defacto operating systems prevail, i.e. the personal and mainframe segments, and the "middle ground" when major growth was predicted but lacked any dominant operating regime.

It was this absence of a single operating system standard that was seen as a constraint on the development of the middle ground or Departmental computing market. The continuation of propriety operating systems fragmenting the market into small machine specific populations that would not attract the software industry to develop applications and hence the application software tends to be limited to that developed by the manufacturer. This means that the computer manufacturers find themselves caught in a vicious spiral with insufficient applications to expand their base and too small a base to attract the independent software industry to develop applications for it.

Recognising this problem, X/OPEN was formed in early 1984 to adopt an Industry Standard Operating System (ISOS) in the "middle ground". At that time, the only credible ISOS appeared to be UNIX, and it is the System V Interface Definition (SVID) that was eventually adopted as the first plank of the Common Applications Environment.

4. When did X/OPEN happen?

As mentioned above, X/OPEN was originally formed in early 1984, though at that time it was wholly European with the initial five members being: Bull, ICL, Siemens, Olivetti, and Nixdorf (in these early days it was called BISON from the company initials).

During 1985 Philips and Ericsson joined, and the first edition of the Portability Guide was published, making the X/OPEN standards available to the public.

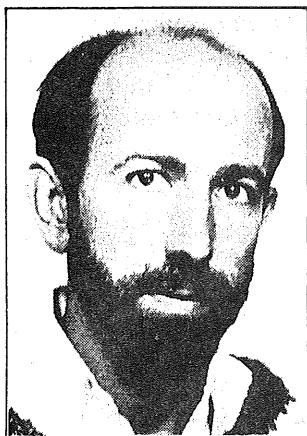
In 1986 X/OPEN gained its American members, Hewlett-Packard, Sperry (Unisys), and DEC, with AT&T joining in January 1987 at the same time as the second edition of the Portability Guide became available.

In three years X/OPEN had evolved from an idea to a practical reality backed by eleven major international information systems suppliers and with wide support from users, software industry, and government.

EUnet

*Peter Houlder
uknet@ukc.ac.uk*

Computing Laboratory, University of Kent



Peter Houlder has been in the Computing Laboratory at the University of Kent for the last 30 months and looked after day to day uknet admin work in the last 18 months of that period.

He graduated in Geography from Kings College, London in 1970 and then spent 9 years in business — dropping out in 1979. He then spent a year touring North, Central, South and Caribbean America, became interested in archaeology and spent three years excavating in Britain and Europe.

Two Masters degrees, the first in Archaeological Sciences and the second in Computer Science, followed in successive years. Maggie in the meantime reduced archaeological funding, so he arrived in 1984 kicking and screaming in the world of Computing. He has since got to quite enjoy it.

He is married with two labradors.

1. Introduction

This is the first of a series of articles giving information about the European UNIX network, EUnet. This particular article contains a short section on the UK network and it is hoped that network administrators in different countries will write later articles, or sections for inclusion in articles.

2. EUnet as part of International Networks

EUnet is the European UNIX network, which started in April 1982 at the European UNIX Systems Users' Group (EUUG) meeting in Paris. EUnet is part of the international group of UNIX based networks, which at present include ACSnet (Australia), USENET (USA), CDAnet (Canada), JUNET (Japan), SDN (Korea) and unnamed network in Israel and New Zealand. Unlike its US predecessor, which splits news and mail as two separate services, EUnet uses the same network for both news and mail. There is some confusion in terminology when referring to USENET. Officially it should be only used only to refer to the North American mail network, but unofficially it tends to be used as a term for all the international UNIX networks. The number of UNIX hosts connected to the international networks listed above varies daily, but the number of unique claimed names at present (27/4/87) stands at 9509. The backbone site for EUnet is **mcvax** in Amsterdam, which has direct links to all the other intercontinental and European backbone sites, along with direct or indirect links to many other non-UNIX based networks. Each

country in EUnet also has its own backbone site: see table below. There are also many important feed sites on the individual national networks, such as **seismo** and **ucl-cs**, which have important links to other networks. All backbone and feed sites must be capable of running UUCP, UNIX to UNIX CoPy, but it is possible for non-UNIX based sites to connect to a UNIX site for mail purposes. All backbone and feed sites must be capable of running UUCP, UNIX to UNIX CoPy, but it is possible for non-UNIX based sites to connect to a UNIX site for mail purposes. All backbone sites and some feed sites provide automatic routing, news feeding and some gatewaying between networks, but only the backbone sites handle registration. Backbone sites are important for several reasons. First they ensure uniqueness of uucp names, as backbone sites should only register unique names. Second they try to ensure protocols are maintained, by rejecting mail or warning sites that indulge in dubious mail practices. Finally they pay the bills to the various carriers and in turn collect the money to pay those bills on a network agreed usage basis.

3. EUnet

At present 16 countries, listed in the table below, are part of the EUnet. This is not strictly true as the Greek and Norwegian sites are only acting as backbones and the Yugoslavian site is not yet on-line. The actual size of any particular network is however difficult to assess because "hosts" may be anything from single-user machines to gateway machines for the internal networks of large multi-user organisations. The other problem is that sites may use more than one name, either because of name aliasing or the use of a local network that is not hidden to the outside world. The terms "site" and "host" are both used to refer to individual mail-handling machines connected directly to EUnet national networks.

EUnet as of 27th April 1987				
Country	Hosts	Names Used	Backbone	E-mail
Austria	19	27	tuvie	tuvie!plank
Belgium	11	29	prlb2	prlb2!ml
Denmark	38	42	diku	krus@diku
Eire	9	10	einode	einodelsimon
Finland	45	45	tut	tut!hmj
France	68	68	inria	inrialdevill
Great Britain	208	248	ukc	uknet@ukc.ac.ukc
Greece	4	8	ariadne	ariadne!kostas
Iceland	1	1	hafro	hafro!gunnar
Italy	25	25	i2unix	i2unix!roby
Netherlands	93	159	mcvax	piet@cwi.nl
Norway	6	7	nuug	kvvax4!franki
Sweden	123	146	enea	enea!ber
Switzerland	29	29	cernvax	dietrich@cernvax
West Germany	107	107	unido	ap@unido
Yugoslavia	1	1	yupiter	yupiter!root
Total	787	952		

3.1 Uknet as part of EUnet

Great Britain started its EUnet links back in 1982, but a fortuitous short term link, via an ex-student, directly to USENET in the USA, meant that its close involvement

really began in 1984. In early 1985 it had some 29 sites all connecting directly or indirectly to the Computing Laboratory at the University of Kent, hereinafter referred to as ukc. The creation and continuation of the network is almost entirely due to Peter Collinson, who had the necessary UNIX know-how and contacts to get the network started. However Sean Leviseur, Richard Hellier and in the last 18 months myself have all helped with support software and day-to-day administrative work. The first sites to join the network were predominately academic or commercial sites with close academic affiliations. Later growth has however been fairly evenly split between academic and commercial sites. In the first three months of the network the number of sites doubled to some 60 sites. By the end of 1985 this number had increased to 80 sites. Since that time the number of sites has grown by a steady 8 sites per month, and now stands at 209 sites (27/4/87). The growth in the network shows no sign of flattening off, so if this continues in the foreseeable future approximately 100 extra sites can be expected to join annually.

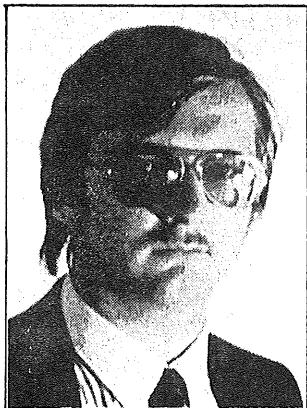
4. Further Reading

An excellent article on international networks called "Notable Computer Networks — John S. Quartermain and Josiah C. Hoskins" appeared in the October 1986 edition of The Communications of the ACM. Some of the above information has been gleaned from this article.

UNIX Clinic

*Nigel Horne
njh@root.co.uk*

ROOT Technical Systems



Nigel Horne has worked solely on UNIX since graduating in 1980 from Westfield College, London (and to a certain amount as an undergraduate as well). He has been involved in UNIX from the early days of "real" UNIX, the days of `seek()`, `roff`, PDP11's (they didn't even have split I+D in those days), keys for typing in the bootstrap, through to today when there are System V, 4.3 BSD, industry standards, and just as much confusion as when it all started.

Nigel is now a Director of Root Technical Systems.

It is hoped that this page becomes a regular feature in future EUUG newsletters. The idea is to start a forum of discussion and trouble shooting, on all aspects of using the UNIX system. Whilst many of the questions may well be slanted towards the beginner, it is hoped that there will be something of interest for just about everyone in the column.

You can send questions to me either via EUUG, by direct mail or even using electronic mail if your machine is connected to EUNET either directly or via another machine. If you want to try sending mail electronically try both of the following commands: if neither of them work, it is unlikely that your machine is connected to EUNET.

mail mcvaxlukc!root44!njh

or

mail njh@root.co.uk

I'm sorry that I can't enter into any discussions about advice given in this column, and any material sent to me by any of the means above will be deemed to be acceptable for publication.

As an introduction I thought I'd cover two questions in one by covering a question that recently showed itself to me. I've slightly doctored it for this example and no names are mentioned to hide certain peoples' identity. The problem manifested itself on a PC/AT look-alike running System V Release 2. Everything was in order except when we came to use the supplied screen editor `vi`. All that the editor did was to print the message `memory fault -- core dumped` and leave the terminal in a strange mode — without echo and the such. The problems here were: why didn't `vi` work, and how do we get the terminal back into a sensible state? Answering the second question is easy. The terminal was left in so called "raw" mode, which meant no echo, no backspace facility, and the system no longer accepted carriage-return as you'd expect. The cure is simple: first type control J. Why? Ah well, UNIX actually takes control J to mean end of input, not carriage return; however it

normally maps one on to t'other, so typing control J just clears any junk characters in the input buffer. After doing this, type

```
stty sane<^J>
```

making sure to use control J again. This brings the terminal and keyboard back into a "sane" state, that is with echo on, carriage return accepted, and so on.

The problem of the core dump? This took some looking for. No other programs on the machine acted in this way, and we began to suspect that our copy of the image of vi on the hard disk was corrupt. In fact it was far simpler, we were using a PC/AT with 512Kb of RAM. vi needs more RAM than this to enable it to run (remember that a fair proportion of the 512Kb is taken up by the UNIX operating system image), and instead of exiting gracefully with a *need more core* message and returning the terminal to a sane state, it just crashed. Solution? Buy more memory.

I hope to hear any questions about UNIX that you may have in the near future. I regret that I cannot answer questions about which hardware to buy, or that I may not cover all the questions I receive, but rest assured that I will try to acknowledge all material I receive.

Review of IEEE Trial-Use Standard Portable Operating System for Computer Environments POSIX+

*Cornelia Boldyreff
...!mcvaxlukc!readingluoseev!corn*

*Department of Electronic and Electrical Engineering
University of Surrey
Guildford, Surrey GU2 5XH*

The draft standard published by the IEEE for comment and criticism was issued in April 1986 with the proviso that its distribution for comment shall not extend beyond one year. In order to facilitate wide-spread distribution, the standard is available from the IEEE and ANSI as well as Wiley-Interscience. Its purpose is to define a standard operating system interface and environment based on the UNIX operating system. Primarily, its focus is the C language operating system interface required to support portable applications at source code level. Similar issues are addressed by the AT&T publication, System V Interface Definition (SVID). The SVID addresses source-level interfaces across AT&T's UNIX System V product; however, unlike the SVID, the POSIX standard is not a specification of a commercial product. The X/OPEN group of UNIX manufacturers has also defined a similar UNIX applications interface, the Common Applications Environment (CAE), based on the SVID, their principal aim being to ensure software portability of UNIX-based commercial products. Like the SVID, CAE is tied to the AT&T UNIX product. X/OPEN has expressed its long term support for the POSIX standard, and AT&T's SVID states that conformance with the IEEE standard will be "strongly considered" after its formal approval.

There are three major components to the standard:

- Definitions — this initial chapter deals with terminology used through the standard, general concepts are described informally, and various symbolically named variables and constants are defined.
- System Interface and Functions — this forms the core of the standard. These chapters define a C Language Binding for Process Primitives and the Process Environment; Files, Directories and File Systems; Input and Output Primitives; Device- and Class-Specific Functions; and Password Security.
- Key Interface issues — Portability; Media Formats; and Error Handling and Recovery.

Currently, the POSIX standard does not address the user interface and associated commands; graphical interfaces; DBMS interfaces; record I/O; or object or binary code portability. Since publication, the P1003 Working Group has formed in addition two new groups addressing the shell and tools interfaces and conformance testing:

- P1003.2 — The shell and tools facilities
- P1003.3 — Verification test specifications

POSIX explicitly does not provide recommendations for an end-user interface; the recently formed POSIX subcommittee is concerned with shell and tools facilities from

the standpoint of syntax and services that an applications programmer might wish to access via the `popen` or `system` function calls. There are other groups concerned with defining a User-Interface for Applications: the X/OPEN group and the ECC ESPRIT PCTE. The latter tests are required because there is a Federal Information Processing Standard targetted for POSIX in the USA.

The IEEE Working Group formulating the POSIX standard includes staff from all the major US computer companies in the UNIX market. Those at a recent meeting included staff from Amdahl, Apollo, AT&T, Charles River, Concurrent Computer Corp, DEC, DG, Gould, IBM, Interactive Systems, H-P, P-E, Sperry, Sun, Tektronix, and TI. UNIX user groups (USENIX, X/OPEN /usr/group) and US government and military users are also represented on the Working Group; and there has been some participation from outside the USA including British members. British companies active in reviewing the POSIX standard include British Airways. In the Acknowledgements concluding the P1003.1 text, over 200 organisations are thanked for their contributions to the Working Group.

The POSIX standard is closely related to two other standards: the 1984 /usr/group Standard and the Draft Proposed ANSI Standard for C formulated by the ANSI X3J11 Committee. The /usr/group Standard work has been subsumed by IEEE P1003's POSIX Standard and ANSI's X3J11 C Standard work. P1003 has left the definition of library functions required for a C implementation in any environment to X3J11; that is, POSIX refers to the C Standard for these. The C standard in turn does not define operating-system-specific functions, leaving these as the province of the POSIX P1003 Standard. There is active liaison between the P1003 committee and the X3J11 committee who over the past few years have developed a good working relations.

This Trial Use Standard is expected to become a full-use IEEE standard and an ANSI standard within two years; ANSI in turn have proposed to ISO a New Work Item based on the P1003.1 effort. Their aim is to facilitate international participation in this work leading to its adoption as an international standard.

In its present form, POSIX does not provide a "functional" specification of a portable operating system independent of any specific language, i.e. in this case C, binding.

POSIX is a much needed effort to standardise an existing applications environment interface based on UNIX systems and complementary to the C standard work. If it could be made to assume this more generalised functional role, then it could well become the basis for related standards work in the area of Open Systems Interconnection.

THIS PAGE INTENTIONALLY LEFT BLANK

Letters to the Editor

Date: 29 Jun 87 12:51:21 +1000 (Mon)
From: greg@gris.oz (Greg Rose)
To: john@moncskermi
Subject: Determination to have the last word

Re: Chris Rusbridge's letter to the editor
Re: The Claytons Unix Programmer

In my defence, I just want to mention that I was talking about *Future needs* and how Unix *Would* solve them. If I at any time said (or wrote) that this was already the state of affairs, it was a slip of the tongue (or digital dislexia, resp.). I also mentioned that some of the setup of menu systems, particularly for system administration, was a significant advance; I like to think that I did not say it was, as yet, acceptable.

In short, I agree with your comments, and it is certainly nice to know that somebody read the article. I might talk again one day.

CASS GROUP

11 Mahogany Crescent
Karama N.T. 5793

FAX 61 89 411 016

G.P.O. Box 866
DARWIN N.T. 5794

00000000000000000000

PHONE: (089) 279 957

00000000000000000000

Mr J Carey
AUUGN Editor
Computer Centre
Monash University
Clayton, VIC 3168

Dear John,

I found the article "A Supplemental Document for AWK" by John W Pierce very interesting. Section 3.8 sought comments regarding output redirection, and as we have had this same type of problem, the following crude methods may give food for thought. Hopefully it will inspire someone to show us neater ways of achieving the same results.

The CASS accounting system consists purely of shell and C shell processes. It does not use C, and it was written after a search throughout the known sources of software in Australia indicated that accounting systems at that time were reworks of Cobol or other languages. The indexing, naming and order of general ledger accounts are determined by the user. Assuming that the user has allocated the code 200600 for telephone expenses, the system provides for 2 files for each account. Thus the telephone account has the files "200600b" and "200600e". The first line of each file contains the number and name separated by a tab, e.g. "200600 Telephone". The second line of the "b" account contains the current balance at \$1, while the "e" account can best be described as a bank statement style account, with each transaction or "entry" including the closing balance, being appended to the file. The number of accounts in a small commercial system would be about 50, thus requiring 100 files to be generated,

OUTPUT EXCEEDING 10 FILES

The task was how to process the chart of accounts through an awk process which produced 2 files as described for each account. The process "split" creates files, but the naming is arbitrary. The following process, which we have dubbed "splitchart" produces the required result.

```

splitchart glchart
#
#splitchart
#
onintr cleanup
if ( $#argv < 1 ) then
    exit
endif
foreach i (`awk ' BEGIN { FS = "      " ; OFS = "      " }'
              { print $1 }' $1`)
egrep $i $1 >! tmp
awk ' BEGIN { FS = "      " ; OFS = "      " } { print $1,$2 }' tmp > $i"b"
awk ' BEGIN { FS = "      " ; OFS = "      " ; zero = "0.00" }'
     { print zero }' tmp >> $i"b"
awk ' BEGIN { FS = "      " ; OFS = "      " } { print $1,$2 }' tmp > $i"e"
end
cleanup:
if ( -e tmp ) then
rm tmp
endif
exit

```

We have adapted this to other operations. For example, an importer allocates a unique inventory number to each item, when preparing customs documents. The customs schedule is awked by a variation of splitter to create a file for each item.

INPUT INVOLVING INDIRECT ACCESS

Sometimes the difficulty lies in not being able to directly access the file through an awk process. For example, the file name may have been generated by an awk process, and it is not the output file that is required, but rather the particular field within the output file which contains the filename.

In our general ledger system, the operator is required to input the account codes to the debit and credit files, followed by some other details. A typical line involving the telephone and bank account would read:

```
"200600    300101    pay    phone    a/c    155.77"
```

with tabs separating each of the 6 fields. That input is output to the file "onejnl". However it is not only the file "onejnl" that is required, but also the files specified in \$1 and \$2 of standard input. To process the account, other files, namely date, sequential ID and the awk file are all marshalled into the file awk1. The file awk1 has already been created when the system is initialised, and has been changed to executable mode.

When the complete input is signalled by CTRL d, the following command is executed.

```
awk '{ print > "onejnl" ; printf "awk -f gljnlawk ndate glcount
%s %s onejnl\nnewline", $1"b",$2"b" > "awk1" }'
```

After execution awk1 contains the command:

```
"awk -f gljnlawk ndate glcount 200600b 300100b onejnl"
```

which it executes.

INDIRECT INPUT/OUTPUT

If output from one awk process is likely to exceed the 10 file limit, intermediate output can be directed to one file for treatment by a subsequent awk process. In the following example the file names (account numbers), which are always located in \$1 are embedded in output which itself is an awk process located in an executable file. The awkprocess1 file contains:

```
BEGIN { FS = "      " ; OFS = "      "
prefix = "awk -f awkprocess2 " ; suffix = "b"
conclude = " | tee tem ; awk -f awkprocess3 tem "
} print prefix ($1 suffix) conclude }
```

Using the telephone account example, one output line of awkprocess1 would read:

```
awk -f awkprocess2 200600b | tee tem; awk -f awkprocess3 tem
Awkprocess2 selects the account balance ($1 of line 2) and performs some calculations, then writes the output to "tem". Awkprocess3 processes each line, directing the output back to each original input file. In this case, file "tem" contains only 1 line. In other applications, tem may contain an indefinite number of command lines, accessing a number of files, and outputting to the same files. Thus while one awkprocess is not permitted to produce 50 lines of output, 50 command lines may each have 1 line of output, because awk recommences counting its output each time it restarts the process.
```

The awk file plawk produces a Profit and Loss Statement from the balances of selected files. The command "awk -f plawk [0-2]*b" will not work from within the C shell program. However the following commands do work because awk1 is a shell process.

```
echo "awk -f plawk [0-2]*b | tee glpltem" > awk1
awk1
```

Yours sincerely,



K Frame
Managing Director
C.A.S.S. Pty Ltd
8 July 1987

AUUG

Membership Categories

Once again a reminder for all “members” of AUUG to check that you are, in fact, a member, and that you still will be for the next two months.

There are 4 membership types, plus a newsletter subscription, any of which might be just right for you.

The membership categories are:

- Institutional Member
- Ordinary Member
- Student Member
- Honorary Life Member

Institutional memberships are primarily intended for university departments, companies, etc. This is a voting membership (one vote), which receives two copies of the newsletter. Institutional members can also delegate 2 representatives to attend AUUG meetings at members rates. AUUG is also keeping track of the licence status of institutional members. If, at some future date, we are able to offer a software tape distribution service, this would be available only to institutional members, whose relevant licences can be verified.

If your institution is not an institutional member, isn't it about time it became one?

Ordinary memberships are for individuals. This is also a voting membership (one vote), which receives a single copy of the newsletter. A primary difference from Institutional Membership is that the benefits of Ordinary Membership apply to the named member only. That is, only the member can obtain discounts on attendance at AUUG meetings, etc, sending a representative isn't permitted.

Are you an AUUG member?

Student Memberships are for full time students at recognised academic institutions. This is a non voting membership which receives a single copy of the newsletter. Otherwise the benefits are as for Ordinary Members.

Honorary Life Memberships are a category that isn't relevant yet. This membership you can't apply for, you must be elected to it. What's more, you must have been a member for at least 5 years before being elected. Since AUUG is only just approaching 3 years old, there is no-one eligible for this membership category yet.

Its also possible to subscribe to the newsletter without being an AUUG member. This saves you nothing financially, that is, the subscription price is the same as the membership dues. However, it might be appropriate for libraries, etc, which simply want copies of AUUGN to help fill their shelves, and have no actual interest in the contents, or the association.

Subscriptions are also available to members who have a need for more copies of AUUGN than their membership provides.

To find out if you are currently really an AUUG member, examine the mailing label of this AUUGN. In the lower right corner you will find information about your current membership status. The first letter is your membership type code, N for regular members, S for students, and I for institutions. Then follows your membership expiration date, in the format exp=MM/YY. The remaining information is for internal use.

Check that your membership isn't about to expire (or worse, hasn't expired already). Ask your colleagues if they received this issue of AUUGN, tell them that if not, it probably means that their membership has lapsed, or perhaps, they were never a member at all! Feel free to copy the membership forms, give one to everyone that you know.

If you want to join AUUG, or renew your membership, you will find forms in this issue of AUUGN. Send the appropriate form (with remittance) to the address indicated on it, and your membership will (re-)commence.

As a service to members, AUUG has arranged to accept payments via credit card. You can use your Bankcard (within Australia only), or your Mastercard by simply completing the authorisation on the application form.

Robert Elz

AUUG Secretary.

AUUG

Application for Ordinary, or Student, Membership Australian UNIX* systems Users' Group.

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

To apply for membership of the AUUG, complete this form, and return it with payment in Australian Dollars, or credit card authorisation, to:

AUUG Membership Secretary
P O Box 366
Kensington NSW 2033
Australia

- Please don't send purchase orders — perhaps your purchasing department will consider this form to be an invoice.
- Foreign applicants please send a bank draft drawn on an Australian bank, or credit card authorisation, and remember to select either surface or air mail.

I, do hereby apply for

- Renewal/New* Membership of the AUUG \$55.00
 Renewal/New* Student Membership \$30.00 (note certification on other side)
 International Surface Mail \$10.00
 International Air Mail \$50.00

Total remitted

AUD\$_____

(cheque, money order, credit card)

* Delete one.

I agree that this membership will be subject to the rules and by-laws of the AUUG as in force from time to time, and that this membership will run for 12 consecutive months commencing on the first day of the month following that during which this application is processed.

Date: ____ / ____

Signed: _____

Tick this box if you wish your name & address withheld from mailing lists made available to vendors.

For our mailing database - please type or print clearly:

Name:

Phone: (bh)

Address:

..... (ah)

Net Address:

Write "Unchanged" if details have not
altered and this is a renewal.

Please charge \$____ to my Bankcard Mastercard.

Account number: Expiry date: ____ / ____.

Name on card: _____ Signed: _____

Office use only:

Chq: bank _____ bsb _____ a/c _____ # _____

Date: ____ / ____ \$ _____ CC type _____ V# _____

Who: _____ Member# _____

Student Member Certification (*to be completed by a member of the academic staff*)

I, certify that
..... (name)
is a full time student at (institution)
and is expected to graduate approximately ____ / ____ .

Title: _____

Signature: _____

AUUG

Application for Institutional Membership Australian UNIX* systems Users' Group.

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

To apply for institutional membership of the AUUG, complete this form, and return it with payment in Australian Dollars, or credit card authorisation, to:

AUUG Membership Secretary
PO Box 366
Kensington NSW 2033
Australia

• Foreign applicants please send a bank draft drawn on an Australian bank, or credit card authorisation, and remember to select either surface or air mail.

..... does hereby apply for

New/Renewal* Institutional Membership of AUUG \$250.00

International Surface Mail \$ 20.00

International Air Mail \$100.00

Total remitted

AUD\$ _____

(cheque, money order, credit card)

* Delete one.

I/We agree that this membership will be subject to the rules and by-laws of the AUUG as in force from time to time, and that this membership will run for 12 consecutive months commencing on the first day of the month following that during which this application is processed.

I/We understand that I/we will receive two copies of the AUUG newsletter, and may send two representatives to AUUG sponsored events at member rates, though I/we will have only one vote in AUUG elections, and other ballots as required.

Date: ____ / ____

Signed: _____

Title: _____

Tick this box if you wish your name & address withheld from mailing lists made available to vendors.

For our mailing database - please type or print clearly:

Administrative contact, and formal representative:

Name: Phone: (bh)

Address: (ah)

..... Net Address:

.....
.....
.....

Write "Unchanged" if details have not
altered and this is a renewal.

Please charge \$____ to my/our Bankcard Mastercard.

Account number: _____ Expiry date: ____ / ____ .

Name on card: _____ Signed: _____

Office use only:

Please complete the other side.

Chq: bank _____ bsb _____ a/c _____ # _____

Date: ____ / ____ \$ _____ CC type _____ V# _____

Who: _____ Member# _____

Please send newsletters to the following addresses:

Name: Phone: (bh)
Address: (ah)

..... Net Address:

.....

.....

Name: Phone: (bh)
Address: (ah)

..... Net Address:

.....

.....

Write "unchanged" if this is a renewal, and details are not to be altered.

Please indicate which Unix licences you hold, and include copies of the title and signature pages of each, if these have not been sent previously.

Note: Recent licences usually revoke earlier ones, please indicate only licences which are current, and indicate any which have been revoked since your last membership form was submitted.

Note: Most binary licensees will have a System III or System V (of one variant or another) binary licence, even if the system supplied by your vendor is based upon V7 or 4BSD. There is no such thing as a BSD binary licence, and V7 binary licences were very rare, and expensive.

- | | |
|--|--|
| <input type="checkbox"/> System V.3 source | <input type="checkbox"/> System V.3 binary |
| <input type="checkbox"/> System V.2 source | <input type="checkbox"/> System V.2 binary |
| <input type="checkbox"/> System V source | <input type="checkbox"/> System V binary |
| <input type="checkbox"/> System III source | <input type="checkbox"/> System III binary |
| <input type="checkbox"/> 4.2 or 4.3 BSD source | |
| <input type="checkbox"/> 4.1 BSD source | |
| <input type="checkbox"/> V7 source | |
| <input type="checkbox"/> Other (<i>Indicate which</i>) | |

AUUG

Application for Newsletter Subscription

Australian UNIX* systems Users' Group.

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

Non members who wish to apply for a subscription to the Australian UNIX systems User Group Newsletter, or members who desire additional subscriptions, should complete this form and return it to:

AUUG Membership Secretary
P O Box 366
Kensington NSW 2033
Australia

- Please don't send purchase orders — perhaps your purchasing department will consider this form to be an invoice.
- Foreign applicants please send a bank draft drawn on an Australian bank, or credit card authorisation, and remember to select either surface or air mail.
- Use multiple copies of this form if copies of AUUGN are to be dispatched to differing addresses.

Please enter / renew my subscription for the Australian UNIX systems User Group Newsletter, as follows:

Name:

Phone: (bh)

Address:

..... (ah)

.....
.....
.....

Net Address:

*Write "Unchanged" if address has
not altered and this is a renewal.*

For each copy requested, I enclose:

- | | |
|---|----------|
| <input type="checkbox"/> Subscription to AUUGN | \$ 55.00 |
| <input type="checkbox"/> International Surface Mail | \$ 10.00 |
| <input type="checkbox"/> International Air Mail | \$ 50.00 |

Copies requested (to above address) _____

Total remitted

AUD\$ _____

(cheque, money order, credit card)

Tick this box if you wish your name & address withheld from mailing lists made available to vendors.

Please charge \$ _____ to my Bankcard Mastercard.

Account number: _____ Expiry date: ____ / ____.

Name on card: _____ Signed: _____

Office use only:

Chq: bank _____ bsb _____ - a/c _____ # _____

Date: ____ / ____ \$ _____ CC type _____ V# _____

Who: _____ Subscr# _____

AUUG

Notification of Change of Address

Australian UNIX* systems Users' Group.

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

If you have changed your mailing address, please complete this form, and return it to:

AUUG Membership Secretary
PO Box 366
Kensington NSW 2033
Australia

Please allow at least 4 weeks for the change of address to take effect.

Old address (or attach a mailing label)

Name: Phone: (bh)

Address: (ah)

.....
.....
.....
.....

Net Address:

New address (leave unaltered details blank)

Name: Phone: (bh)

Address: (ah)

.....
.....
.....
.....

Net Address:

Office use only:

Date: ____ / ____ / ____

Who: _____

Memb# _____