INDRE BY-TERMINALEN
ved Københavns Universitet
Studiestræde 6 over gården
DK-1455 København K
Telefon 01 - 12 01 15

# EUUG

# EUROPEAN UNIX USER GROUP NEWSLETTER

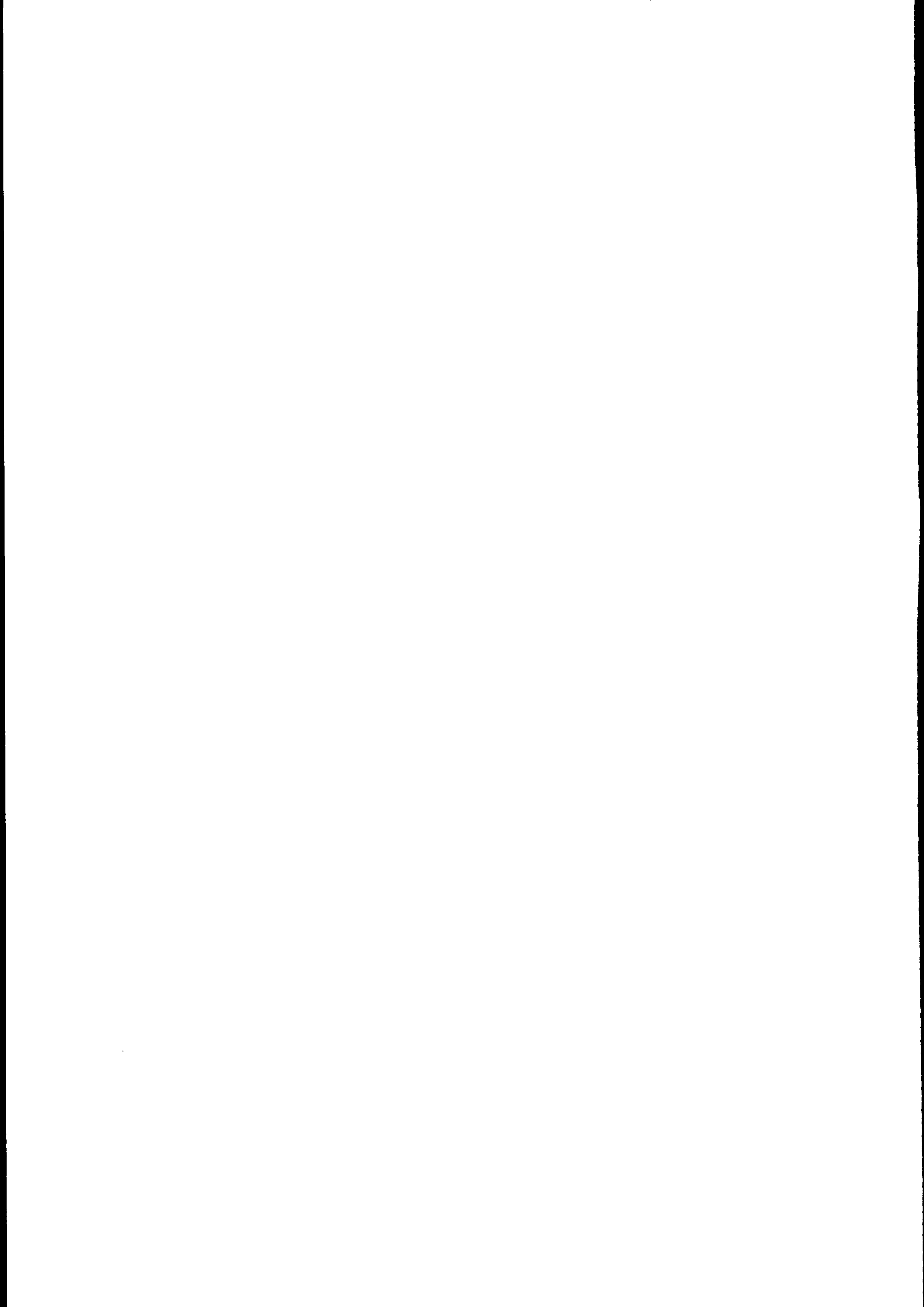
Volume 2, No. 4
**WINTER 1982**

## CONTENTS

| | |
|---|---|
| Editorial | 1 |
| Leeds Report | 2 |
| Design and Structure of an Open Distributed Operating System | 20 |
| Alice | 22 |
| Benchmarks | 29 |
| Software catalogue | 30 |
| Henderson's SECD Machine | 33 |
| Uniflex Evaluation | 40 |
| AT & T Licences | 45 |
| Dutch Unix Bulletin | 47 |
| UNIX for the STD Bus | 49 |
| Prolog | 52 |
| C.: Toward a Concise Syntactic Description | 54 |
| Proposed changes to C | 70 |
| Formatting in C | 73 |
| C Style and Coding Standards | 86 |
| Unix - like system Standards | 107 |
| Ada from York | 112 |
| Losing the Sticky Bit | 125 |
| Onyx | 129 |
| Abstracts | 132 |
| Letters | 136 |
| Press Clippings | 146 |

This document may contain information covered by one or more licences, copyrights and non-disclosure agreements. Circulation of this document is restricted to holders of a licence for the UNIX* software system from Western Electric. Such licence holders may reproduce this document for uses in conformity with their UNIX licence. All other circulation or reproduction is prohibited.

* UNIX is a Trademark of Bell Laboratories.



## Editorial

Well, this is a good time to get rid of all the dead wood that has collected on my desk(s). This is the last Newsletter of '82, and with a bit of luck, we will actually make 4 Newsletters in a 12 month period. All the backlog is in this one, so if you send me a letter or article, I'll stick a cover on it and it will be the next Newsletter.

There are a number of diverse articles here, the report on the excellent Leeds Meeting, some interesting pieces from outside the U.K., and a number of views on Languages.

Things are changing. The EUUG now has a permanent Secretary to field all those enquiries, and to distribute things on time. The address is:

Mrs. Gibbons
European UNIX User Group
Owles Hall
Buntingford
Herts. SG9 9PL
United Kingdom

Tel: Royston (0763) 73039

The other thing to note is that I am moving, so the address for the sacks of articles will be, from 1st January 1983:

Stichting Mathematisch Centrum
Kruislaan 413
1098SJ Amsterdam
The Netherlands

Tel: Amsterdam (020) 5929333

EUNET address . . . . . .!mcvax!jim

No, McVax is not a Scottish machine. Next issue will reveal all about the EUNET, how you can send me articles on it, how useful it is, and where to get it. Network mail is now the preferred means of contact.

The next EUUG Binge is in Bonn, 11th to 13 April 1983. Contact Mrs. Gibbons for details.

Just had a phone call from Stan Meachin, 0734-342666, looking for anyone using MICROSIM. Any takers?

So, all for now. Hope you have/had a pleasant Festive Season. Roll on Vol. 3.

## EUUG Group meeting
## University of Leeds, 6/8 September 1982

Peter Collinson
Secretary

### Day 1 - Applications

The main theme of the first day of the conference was applications on UNIX.* The day was open to all comers and a successful attempt was made to eliminate all the vendor presentations which have made previous conferences so uninteresting (to me anyway). The introduction was done by Emrys Jones.

Item 1: 11.08am John Wilson, Logica

The strengths of UNIX

John had been asked to give a brief introduction to UNIX aimed at those who were there to find out about it.

Item 2: 11.28am John Saunderson, Root Computers

Where is System III taking UNIX?

Root Computers are a case of °mv CDS Root'. John started by talking about the importance of the Anti-Trust decision for AT&T, see Otis Wilson's talk below. He isolated three market areas where UNIX System III is important.

First, the program development area, where he mentioned °make' and °sccs' as important tools. Many large companies are looking at C as a development language. Another important factor for sites with IBM machines is the RJE emulation software.

Secondly, in the area of office automation systems, UNIX provides good mechanisms for communications, networking and electronic mail. Also, System III provides much better system accounting than was previously available.

The third area is in the 16/32 bit micro market because UNIX is available on a wide range of micros.

John ended his talk with some cautions: he feels that standards are important for UNIX and some thought needs to be given to the areas of file locking and system security.

* UNIX is a Trademark of Bell Laboratories.

Item 3: 11.47am

Tom Leonard, Precision Software

The Marriage of C and CIS Cobol

Precision Software use a Bleasdale machine running Xenix to develop packages for UNIX. The company sells software to distributors and not to end users. In order to give a distributor the ability to tailor the product to meet the needs of the end users, it is desirable that packages are written in Cobol because the commercial market place knows about it.

In addition Cobol has a number of advantages, these include: ease of implementation, good I/O, indexed files, arithmetic to at least 18 digits (this is important because packages often need 13 significant digits), good data formatting and portability across many machines. CIS Cobol now comes in two flavours: an interpretive system and a compiler (Level 2).

The company uses C to overcome the restrictions of Cobol and insert C routines into Cobol by calling them via the CALL verb, which is designed to call external functions and routines using a special data section for linkage. C is used to give access to shared files and also to create asynchronous processes.

-- Lunch --

The afternoon session was chaired by Eddie Bleasdale.

Item 4: 2.30pm

Otis Wilson, AT&T

AT&T and UNIX

Otis started by talking about the history of the Anti-trust decision which is altering AT&T, and allowing them to sell software.

On January 8th, 1982, the US Justice department and AT&T reached agreement. On January 15th, the jurisdiction of the case was assigned to Judge Greene and 6 months was allowed for submissions by the public. On August 11th, the Judge issued an opinion which broadly agreed with the way AT&T wanted to do things; and on August 24th, the Judge approved the agreement between AT&T and the Justice department. At the end of 6 months AT&T have to submit a plan for the re-organisation of the company and at the end of an 18 month period, the company must have divested itself of the 32 telephone companies which it wants to get rid of. So, the company is currently going through a planning stage and consequently all the details concerning the sale and licencing of UNIX are not clear.

Otis went on to give the commercial licence fees for various AT&T software products. The numbers below are dollars.

| Software | Initial CPU | Additional CPU | Customers CPU's | Time + Sharing |
|---|---|---|---|---|
| Mini UNIX | 12,000 | 4,000 | | |
| UNIX | 20,000 | 6,700 | 8,400 | 10% AC |
| PWB | 30,000 | 10,000 | 12,000 | 10% AC |
| UNIX V7 | 28,000 | 9,400 | 11,700* | 10% AC |
| UNIX 32V | 40,000 | 15,000 | 18,000 | 10% AC |
| UNIX-SIII | 43,000 | 16,000 | * | 10% AC |
| UNIX/1000 utilities | 30,000 | 10,000 | 12,000 | 10% AC |
| UNIX SYSTEM TSS | 100,000 | 20,000 | | |

* means refer to the binary schedule.
\+ AC means Access charges, so you pay a royalty to AT&T

If a customer wishes to sell binary versions of System III, there is a one time payment of $25,000 which is non-returnable and non-creditable. Then the following table applies:-

| User Capacity per CPU | Cost in $ / CPU depending on Cumulative total of customer fees paid | | |
|---|---|---|---|
| | $0-$1 million | $1-$2.5 million | "$2.5 million |
| 1 | 100 | 70 | 40 |
| 2-16 | 250 | 125 | 50 |
| 17-32 | 1000 | 750 | 500 |
| 33-64 | 3500 | 3500 | 3500 |
| over 64 | 7000 | 7000 | 7000 |

Otis showed the well worn slide containing the magic words: How we licence - as is, no maintenance, no warranties, no patent indemnification, no trial period and payment in advance. He said that he hoped that it was the last time that he had to show that particular slide - so read into that what you like!

He said that the staff in the licencing division had been increased significantly and consequently it was much quicker to get licences: the customer should allow 2 to 4 weeks for AT&T to prepare the agreement after the receipt of the written request. After the agreement and the money has been received by AT&T, it takes 1 day to get the agreement executed and then you should allow 1 to 5 days for delivery of the software - I guess this will take a bit longer in Europe. (My experience is that they ship stuff to the UK using air freight - so allow an extra f100 or so to get the box through customs).

Otis showed a slide of UNIX licencing activities as of the 1st June 1982:

Licenses

| Software | Commercial | Educational | Government | Total |
|---|---|---|---|---|
| Mini unix | 6 | 119 | 0 | 125 |
| UNIX V6 | 90 | 355 | 54 | 499 |
| PWB/UNIX | 46 | 59 | 75 | 180 |
| UNIX - V7 | 133 | 313 | 64 | 510 |
| UNIX - 32V | 71 | 156 | 25 | 252 |
| UNIX - S III | 84 | 0 | 4 | 88 |
| Totals | 430 | 1002 | 222 | 1654 |

Installations

| Software | Commercial | Educational | Government | Total |
|---|---|---|---|---|
| Mini unix | 8 | 358 | 0 | 366 |
| UNIX V6 | 167 | 946 | 135 | 1248 |
| PWB/UNIX | 127 | 223 | 92 | 442 |
| UNIX - V7 | 209 | 915 | 78 | 1202 |
| UNIX - 32V | 105 | 294 | 29 | 428 |
| UNIX - S III | 107 | 0 | 4 | 111 |
| Totals | 723 | 2736 | 338 | 3797 |

Otis went on to make an announcement of the Educational licencing policy for software from AT&T. All educational licences in future will be granted at a System III level, so if you ask for a licence for UNIX V7 - you will get a System III licence but UNIX V7 software. The licences are available to all qualifying institutions whether in the USA or elsewhere. Licences are granted on a per CPU basis and will cost $400 for all software packages. The System III educational licence costs $800 for an initial CPU which includes 1 set of software and documentation. Additional CPU's cost $400 for the agreement to be serviced, and an extra $400 for a software distribution. Institutions who are already holding licences may upgrade all currently licenced CPUs along with their initial CPU request for the $800 fee. So, make sure that you include all your currently licenced CPUs on your request for a System III licence.

Administrative fees for System III are $16,000 for an initial CPU and $5,400 for each additional CPU. Full upgrade credit is given for current administrative CPUs.

For all licencing for System III write to:

Otis Wilson,
Technology Licensing,
P.O. Box 25,000
Greensboro, NC 27408, USA

Phone: 919-697-2078
919-697-6530

In reply to a question, Otis said that you still have to have a licence to get the System III manuals. He also said that it was possible to licence

the C compiler separately: it costs $4000 + $2000 per/CPU, note that this is lots more than a UNIX licence. Also, if you have sub-licencing rights, AT&T can supply a list of which software may be supplied to a customer on a sub-licenced CPU - this includes some required source files.

-- Tea --

Item 5: 4.02pm Emrys Jones, Cgram Software

Programming applications in C

Emrys talked about the way his company is writing software in C, his systems are aimed at small processors. The objectives of the software are:

1) Fast response time. This means that programs should be small to minimise system overheads. It is also important to minimise file opening time. Code must be efficient and the system must utilise economical disc data base systems.

2) Portability of the software and transportability of data files between machines. The latter requirement means that some machines need byte ordering code to be built in.

3) An elegant user interface. Screens of data must be built in a predictable fashion (unlike some systems where the cursor darts about the screen, updating as it goes); good output buffering is required and file I/O delays must occur predictably.

4) Comprehensive security. Each user has a security profile file which is accessed at login time, this data restricts access to the parts of the file system which they have a right to see.

5) Fast development of individual applications.

6) Ease of modification.

To do this, their software required the following tools. First, a file infrastructure has been implemented which uses B-trees to contain keyed data files. It is possible to lock records or lists of data in a file. To give a locking capability to UNIX, a small device driver containing a semaphore system was added to the UNIX kernel. The file structure can be checked to be internally consistent.

The programs are mostly °data driven' from tables of constants. The tables are C structures which are pre-loaded using the standard C initialisation syntax. They found that there was a need to perform type checking on the contents of the structures, because the compiler is not good at recognising errors. To do this, they have developed a tool which checks the synchronisation of data types inside the structures.

Item 6: 4.23pm Peter Osborne, Redwood Software

Word processing in C

Peter described the UNIPLEX system which is a word processing and menu selection system. The software is written in C and this was chosen because it can run on a wide variety of computers and operating systems.

The UNIPLEX program is driven by a text data file and presents a series of menus to the user. The script for the data file can access existing UNIX facilities as well as act as a word processor.

END OF DAY 1

**Day 2**

The day was chaired by Roger Boyle, from Leeds University.

Item 7: 9.24am Evan Adams, Amdahl Corp

UTS

UTS is a UNIX sub-system running on the Amdahl 470. This processor is a plug compatible IBM 370.

In 1977, Amdahl employed a lot of graduates from Berkeley who were fairly reluctant to use IBM existing operating systems. Their managers decided that they could run UNIX, but higher management wouldn't buy a PDP11 because "this company manufactures computers". A start was made in porting UNIX to one of their machines, and Amdahl got a V7 tape, but the project was cancelled. In 1978, Princeton students ported UNIX onto an IBM machine. This lead to a re-emergence of the idea of Amdahl attempting to port UNIX.

UTS does not run on the bare machine but runs on top of the VM operating system on the 470. VM supports °virtual' machines and it is possible to run different operating systems concurrently. However, there was a bootstrap problem. It was necessary to use an existing time-sharing operating system (CMS) running on a 470 as a base for the development because the maxim of "we make computers" meant that they couldn't start from a UNIX machine. Evan felt that porting UNIX from a UNIX machine is much easier.

The first problem was to get a compiler for C. Bell had a compiler called C370 which is a subset of C and generated code for CMS. It was necessary to write programs to convert from CMS internal formats to UNIX formats and also to write an IBM assembler and loader. Other required changes to UNIX are in the area of I/O handling and memory management. The system also had to be made to handle the standard IBM full screen, half duplex terminal. A lot of work has gone into making a better C compiler.

In early 1979, a V6 UNIX implementation was running on the machine. The system was slow and unreliable; this was mainly due to the VM overhead in handling priviledged instructions. It was necessary to recode parts of the kernel so that device interrupts could only occur in certain well defined sections rather than spending lots of time changing priority in the kernel via the spl() routine calls.

In November 1979, the V7 tape arrived and a port from the old to the new system done. This was much easier than the original port from CMS to UNIX. The V7 test system was running by February 1980, the use of VM meaning that it was possible to run both systems in the same machine for testing purposes.

To improve performance, the disc block size was increased from 512 bytes to 4096. In the current system, the VM overhead is 25% to 30%. /tmp is a core file to get better performance. Evan mentioned that the system could support in the region of 150 users. There are 20-25 sites running UTS.

Amdahl are currently thinking of running UNIX on a bare machine.

Item 8: 10.00am David Tilbrook, Computer Systems

Alice

Alice is the "You kin get anything y'want at Alice's restaurant" menu system but Alice doesn't stand for "another language involving C enhancements".

Alice is a software tool running under UNIX. It provides facilities for developing interactive application packages with a simple and consistent user interface. Its primary use is for applications for the novice or casual user, which can be easily extended or modified in parallel with the user's needs or experience.

The system is script driven and can contain specifications for four basic questions: the user can be asked for a value; can be asked to answer °yes' or °no' where yes is the default; can be asked to answer °no' or °yes' where no is the default; or the system can pause and/or paginate the screen. Responses from the user are consistent and allow the user to ask for an explanation of the command (supplied in the script) or for a list of available commands.

The system can be used to generate management systems, special user systems, or general office systems. All of which can be used by personel with little or no UNIX experience. It also allows the knowledgeable user to handle a large variety of problems involving data bases, complicated procedures that require user interaction (i.e. creating new users) or as front ends to sets of related applications.

(Thanks to David for the blurb on which this section is based).

-- Coffee --

Item 9: 11.10am John Collins, Root Computers

So what's different about System III?

Root Computers are selling and supporting UNIX System III in the UK, they can supply binary licences on the PDP11 and VAX ranges and can also sell turn-key systems. Their current developments in UNIX consist of: security enhancements, ISAM package, a relational database, BASIC, additional peripheral support, word processing software and accounting programs (sales/purchase ledger).

What's new on System III? SCCS - the source code control system; extended °make'; first-in first-out devices - which are sometimes called °named pipes'; non-wait I/O; mm macros for nroff; system accounting software; C compiler; EFL - extended Fortran language (the manual is missing); °cpio', a new tape driving program; and °chown', which can be used by anyone.

What's missing? Multiplexed I/O devices, the ms macros for nroff, the °at' system, °refer', °struct', secret mail and °learn'.

The things which don't work are: the standalone system, which allows certain utilities to be run in the bare machine; °volcopy' for backup of disc volumes; system activity reporting; handling the UNIBUS map on the 11/44; there are problems with the CLOCAL bit and the floating point interpreter on separate I/D space machines; and there are problems with non wait I/O. There

are compatibility problems with V7 UNIX in the area of the terminal handler, the internal structures used for stty/gtty are different. The system is supplied in °cpio' format and you need a °cpio' binary to read it.

Root Computers are putting work into: Berkeley Pascal, user overlaying for C an F77 processes, file locking, file quota control, RX02/RX03 floppy disc driver, RM02/RM03 disc driver, the standalone software and an enhanced spooler.

Item 10: 11.56am — Chris Miller, Leeds University

Experiences with Eunice

Eunice is a system which emulates UNIX under VAX/VMS. The system was written and originally supplied by SRI international and $500/cpu. There was no support for the system under the original licence agreement and there is no Eunice source in the basic system. Currently, Eunice is supplied by the Wollongong group at $5000/cpu including support for the first year, after that a maintenance charge is required to get support.

The system is almost Berkeley 4.1BSD running under VMS versions 2 & 3, it has the C shell, APL, Lisp, vi/ex, vtroff and the standard 32V utilities. The file system is identical with the VMS file hierarchy, files created under Eunice can be read from VMS and vice versa. However, file names are restricted by the VMS limitations.

The system can create and run either VMS binaries which can be executed on VMS AND Eunice because the object files contain the usually shared libraries; or you can create UNIX binaries which are only runnable under Eunice.

There are some problem areas. Eunice uses the VMS device drivers, so some ioctl calls work and some don't. The filename restrictions of VMS are a pain. The system runs fairly slowly and tends to be less robust than most UNIX systems running on the bare machine.

Conclusion: if you are forced to run VMS, Eunice is better than nothing.

Item 11: 12.19pm — Adrian King, Logica

UNITY

UNITY is another system which emulates UNIX under VAX/VMS and is a product of Human Computing Resources. The system emulates UNIX completely using no VMS privileges, it uses the VMS device drivers and is installable onto any VMS system. To load the system, the user logs into VMS and from there calls a sub-system which is UNITY.

There are several problems in the emulation of UNIX under VMS. First, it is necessary to emulate the UNIX process hierarchy. This is hard on VMS because VMS only has detached processes with no notion of the parent/child relationship and also, VMS has no °fork' primitive. Although VMS has a structured file system, it has no °root' directory and the file naming conventions are different.

Each user on the UNITY system gets a process manager which controls

processes using message passing as a means of communication. The °fork' primitive is done as follows:

1) Process A creates a forkfile
2) Process A sends a message to the process manager saying °fork'
3) The process manager starts a seed process
4) The seed process reads the forkfile
5) and tells the process manager that all is OK
6) Process A continues.

There are performance problems with fork/exec due to this mechanism.

The file management allows the handling of various VMS file organisations and levels of directories and it copes with the handing of the sharing of files opened by multiple processes. UNIX file links cannot be emulated. However, UNITY does some name mapping so that UNIX file names are allowed. This means that you get some strange VMS names, but this seems much better than the Eunice approach. UNIX files are 512 byte fixed length record files on VMS and it is possible to create VMS typed files from inside UNITY.

Adrian finished with some reasons as to why you should run UNITY. VMS has a lot of packages for CAD/CAM; DEC engineers are happier; and for some sites, the idea of replacing VMS by an unknown system is a bold step.

UNITY has a Beta test release in August and the first production release will be on October 1st. This will be a V7 system, the System III additions and support for Berkeley utilities will be added slowly.

-- Lunch --

Item 12: 2.19pm Mike O'Carroll, MSU, Leeds University

The Micro Systems Unit development facilities

The Micro systems Unit is responsible for courses for undergraduate teaching on micro processors. It provides lab facilities for student projects and supplies advice and a pool of equipment for research staff in all departments of the university.

The lab is based around a PDP11/44 which has 1.5Mbytes of memory, a 134Mb Winchester, a 67Mb SMD disc and 2 * 26Mb RK07's. There are 32 serial ports. The software is UNIX V6 and runs the Whitesmith's cross compilers for Pascal and C to generate code for the PDP11, the 8080/Z80 and the M68000. Leeds have written some down-line loading code.

A number of workstations are attached to the system, a workstation can be a simple vdu or a local micro computer. The micro contains a PROM which allows it to work in either standalone mode or to connect the VDU to the PDP11 as if it were a terminal. It is also possible to down-line load the workstation from the host. The standard work station is a S100 based micro computer, usually the Z80. The Z80 has two serial ports, one connected to a VDU and the other to the host machine. The standard hardware package has 64Kbytes of memory, a digital/analogue I/O card (Cromemco), a counter/timer, and some PROM based software.

The non-disc based system for the workstation has a local monitor which has a boot-strap and elementary utilities. It emulates CP/M console handling

and supplies a run-time environment for down-loaded programs.

The disc based system uses a standard CP/M operating system and has a disc resident linker which allows the workstation to be used as a normal terminal development system.

For further details, contact Mike O'Carroll.

Item 13: 3.04pm Euug Ctte

EUUG Business

There were several items of news on the User group. First, Emrys Jones has been elected Chairman of the Group. The post of Treasurer falls vacant.

Secondly, AT&T has informed the group that it can no longer be called the °European UNIX User Group' but must change its name to the °European UNIX Systems User Group'. However, the old acronym can remain - so we can still be °EUUG', pronounced °eee double-you gee' in English, in Dutch it's something else again.

Forthcoming meetings: the next Spring meeting will very probably be in Bonn but no confirmation has been received about this. Teus Hagen will look into it. The September 1983 meeting will be in Trinity College, Dublin - this is definite.

There then followed a discussion about standards and the need for them. I'm afraid I took no notes - apart from the quote from Bruce Anderson - "UNIX used to be for space cadets and now it's for lorry drivers". However, there was a clear difference in opinion and ideas between those who were interested wanted a standard so that they could sell more UNIXes and UNIX applications programs and those who merely wanted a reference standard so they would know what was involved in porting the software.

-- Tea and bickies --

Item 14: 4.05pm Berkley Tague, Bell Labs

UNIX Operating Systems development: Directions and standardisation

Berkley gave a very interesting talk which shed quite a lot of light on what is happening to UNIX inside Bell. He is responsible for looking at where the system should go inside Bell, his customers are internal to AT&T.

At present, there are four development directions with which he is concerned. These are networking, multiprocesser UNIX, file system/data base management and language development.

On the networking front, there are currently many different ways to interconnect machines and there are several possibly different sorts of interconnections which might be needed. For instance, UNIX to UNIX connections may be different from connections between UNIX and hosts running other operating systems. Also, the connection of terminals to UNIX is likely to alter as terminals get smarter. In fact, terminal connections might begin to look like either UNIX/UNIX or UNIX/other host connections. There are several networking

issues which require solutions. There is a need for a media independent applications interface, it is not clear to what extent it is desirable for the file and device access to be integrated with the inter-process communication. Bell are currently looking at the Berkeley IPC definition (because it's there). Local area network protocols are a hot issue in the US, as are the global protocols; X25 is starting to be used.

For multi-processor UNIX, the issues are several. Should systems be loosely or closely coupled? Are we multi-processing for reliability or capacity? Is multiprocessing on a local area network feasible? There is a lot of interest in single board computers on a common backplane where the backplane supports more than one processor.

The questions in the area of file systems and data base management are as diverse. Should use be made of a DBM sub-system or should the file system just be extended so that existing tools (awk and sed) can be used? If the DMB sub-system approach is adopted, then which DBM? There are lots of ad-hoc DBM systems already developed. If the file system approach is adopted, then the file system will need alteration to provide better robustness, some backup and journalisation, some locking features, concurrency control and record management. There is also a need for transaction processing. A better IPC is needed and the scheduler needs to be able to be controlled to allow priority for certain users.

Languages - well, C is unlikely to be altered radically in the future. Bell are looking at the standard °traditional' languages: F77 (need a good compiler), BASIC and COBOL. They are also looking at languages with stronger typing than C, e.g. Modula, Euclid and Concurrent Pascal. They are also looking at ADA.

Berkley then talked about the Bell Lab's view of research and development inside Bell. This is called the °Bell Labs technology transfer model'. There are three levels; first, the Research group, which has a very long time scale (like 5 to 7 years) for generating anything. The second level is °Applied Research', this is created in order to get things out of the research labs, the time scale of projects is 2 to 3 years but there are no very hard deadlines. Thirdly, the Development group, which is funded by the manufacturing arm of the company and runs using fixed time scales with one year milestones. Bell feel that University work augments research in Bell Labs, the universities act as a cutting edge for new ideas but there are severe technology transfer problems because universities produce °bread-boards' not products.

Berkely's group is also looking at the standardisation issue. They think the goals of standardisation are two fold. First, there should be application code compatability and portability; and secondly, there should be a machine independence definition, so that the system is independent of particular hardware vendors. The approach being adopted is °certification', i.e. a suite of programs is written which test whether a system is °standard'. The programs will test subsets of the UNIX system.

In summary, Berkely said that Bell and the rest of the UNIX user community have common interests in the future developments of the system. Bell needs university research and wants to know about user requirements. Bell want to co-operate on standardisation.

END OF DAY 2

During the day, there was a °competition' on the black board which asked for the most useless new switch for °ls', here are the suggestions for

answers:

1) A switch which prints the files which °rm' can't remove (this is marginally useful).

2) A switch which prints the nulls after the file names.

3) A switch which prints all the file names backwards (so we can find a use for °rev').

4) A switch (preferably the first letter of the arabic alphabet) which prints times in solar time (see ctime(3)).

5) A switch to invoke /usr/games/ching for explanatory diagnostics.

6) A switch which causes all output to be suppressed and returns a status of zero.

7) A switch to delete directories and leave the files hanging about - this switch exists on the Eunice system.

8) A switch to switch off all other switches (just try saying that when you're drunk!)

9) A switch which does nothing but is extremely well documented.

Any more ideas? Send them to the Newsletter Editor.

**Day 3**

The last day of the meeting was mostly about Distributed UNIX and related topics. The day was chaired by Chris Pinches from Leeds University.

Item 15: 9.15am Ian Wand and Andy Wellings, University of York

ADA and PULSE

PULSE stands for Personal UNIX Like Systems Environment and is an SERC funded project to develop a distributed UNIX-like environment.

York also want to investigate methods of distributing systems and have chosen the 'personal' computer approach where a number of small machines are connected together by a network. For the approach, each personal computer has its own CPU, disc and network connection. Currently, the processors being used are PDP LSI 11/23's. The network is a Cambridge ring and supports a global file server for shared data. The system is coded in Ada.

Each Pulse machine will provide a kernel for inter-process communication, virtual memory management and Ada task management. The kernel is a message passing system with transparent access to remote resources. The idea is to have a minimum set of kernel facilities and utilise user tasks for anything complicated. The Ada tasking model replaces the UNIX process model, this means all tasks have a shared memory and all tasks in the machine compete equally for resources. The kernel does not contain any file system access.

The interprocess communication system is based on the Carnegie-Mellon University IPC for UNIX and Spice. There is uniform access to user/kernel resources, whether they are local or remote. Access is controlled by capabilities, and these capabilities can be passed from one task to another.

The file system is distributed. Each pulse processor contains a local file server which can be connected to a global server. The aims of the file server project are: first, to generate a global hierarchical file store where each file has an absolute pathname. Secondly, to cope with file replication and last, to allow extensibility.

For more details of the project see the document "Distributed UNIX project 1981", by A.J. Wellings, I.C. Wand & G.M. Tomlinson; York Computer Science Report No. 47.

The second part of the talk concentrated on the York Ada compiler. There are several problems with Ada: it is difficult to write a sensible interrupt handler using tasks; there is a high tasking overhead in Ada; there are high consistency checking overheads; there is no controlled way of cheating the strong typing; there are no unsigned integers. The language is complex to learn and use and the implementation is large and slow.

Despite all this, the preliminary version of the York Ada compiler will be available on the 15th October, this will be released to educational establishments and UK research council labs only. See elsewhere in this newsletter for a form to request the distribution.

Item 16: 10.00am Keith Bennett, Keele University

The Keele Distributed File Store

The objectives of the Keele project are: to have a single name space for all files on the system; to have a consistent client view of the file store - so that the client always sees all the file store even though some of it may not be present; and there should be an ability to have the sharing of files and protection.

Things that were NOT objectives of the project are: distributed UNIX; issues of security; fault tolerance and recovery; data base management systems; general purpose operating systems; and the migration of processes between machines.

What has been implemented is a single network wide tree (with no links). The tree is present for all users. However, the actual files in the tree are not necessarily present on each user's file store even though the names are. The file store structure is replicated on a per disc volume basis, so each disc volume in the file store contains a copy of the structure of the tree.

The main problem with the structure is file replication, the system copes automatically with updating of different copies of the same file. There will be a master file somewhere on the system and a list of replication addresses. The other problem is the consistency of the entire file store. To maintain consistency, each file is given a time stamp which is actually a version number. It is then possible to locate and inspect all on-line copies of a file; compare the time stamps and update files. If a master file is deleted, then an 'assassin' is left which causes copies to be deleted when the volumes come on line.

-- Coffee --

Item 17: 11.03am Andy Tanenbaum, Vrije University

Monix and Amoeba

Amoeba is a fully fledged distributed operating system, it is an operating system aimed at research into distributed operating systems for local area networks. Monix is a new UNIX-like system for real time operations running on personal computers.

Why bother to rewrite UNIX? There are the academic aesthetic reasons; Andy wanted a well-structured UNIX kernel without goto's, side effects, splx, global variables and a big monolithic system. Also, it is not possible to teach the structure of UNIX V7 and systems which came after it. Finally, UNIX is not a real time system and Andy needed the response which you can get from such a system.

Andy has written something on his talk which appears elsewhere in the newsletter.

Item 18: 12.05pm Colin Low, QMC

The Network Shell

Colin wanted to connect several UNIX processors together. The main requirement was for a simple user interface without complicated procedures to learn. The system should be consistent with the overall UNIX design philosophy. It should be simple to install, capable of performing what most people want and should involve no modifications to the UNIX kernel.

The idea was to extend the shell to support some cross machine features, i.e. to allow process invocation and argument passing; the `, ¨, and | operators; and to allow current working directory access.

Files are addressed by °connecting' all the trees of the separate tree structures together and allowing some of the names in the tree to represent the addresses of remote machines.

The operation of the system starts with a network login program. This keeps forking to set up connections to each machine and uses an initial connection protocol to invoke a remote login process and then the shell in the normal way. The network shell thus retains connections opened as it goes along and can direct shell commands towards a shell on any connected system. The net shell functions as a switch, passing user's input and deciding which system a given expression refers to. The shell can invoke °glue' processes in any system, these processes are used to communicate between processes running on a machine and the network. There are two of these processes: °netfile', which communicates between a file and the network (or vice versa) and is used to deal with ¨ and `; and °netpipe', which connects the network to a pipe (or vice versa) and is used to deal with the | operator.

All UNIX commands work in the normal way across the network. The main problem is that copying files between machines has to be done by the use of data stream utilities. So, to copy a file you have to say:

cat file ¨ /70/mnt/foo

rather than

cp file /70/mnt/foo

-- Lunch --

Item 19: 2.17pm John Parkinson, Apollo

The Apollo Domain System

The DOMAIN (Distributed Operating Multi Access Interactive Network) is the world's first commercially available distributed operating system. The system consists of several workstations connected by a local area network.

Each workstation consists of (minimal):

CPU 2*M68000 (two because one is used for memory management)
512Kbytes of EEC memory
A4 size bit map display (can be colour)
Display manager with 128Kbytes of memory
33Mb winchester
1 Mbyte floppy disc drive
Block multiplexer (controls network and disc access)
Optional multibus controller
3 * V24/RS232 serial I/O ports

The system costs f23,000 with a disc and f17,000 without one.

Each node is uniquely identified by a number inserted at the manufacturing stage. The network is °home-grown', and is a single token ring network with a 12Mbits/second data rate. The data packet size is 1K bytes, which is the same as the internal disc block size. The network is fast enough to support paging and disc-less nodes do work.

Item 20: 3.02pm — Brian Randell, Newcastle University

The Newcastle Connection

The talk was sub-titled "UNIXes of the World Unite" and described a mechanism for connecting several UNIX systems together.

The Newcastle Connection is a software sub-system which can be added to a set of physically interconnected UNIX or UNIX-lookalike systems to construct a distributed system. UNIX-lookalike means any system which supports UNIX V7 system calls. The resultant system is functionally equivalent to a conventional single processor UNIX system. The Newcastle Connection allows file access, device usage, I/O re-direction, interprocess communication, change directory, mail and remote execution, all to take place on and between several machines. It does this by conceptually joining the file systems of the machines into one huge tree, so that every machine can address every other machine in a consistent manner.

One particularly neat idea with this scheme is the use of the chroot system call to move the root of the file system °down one', so that all remote addresses are of the form:

/../\`machine id¨/\`destination on remote machine¨

This means that your local machine is conceptually °under' a vast invisible tree which connects all the UNIX systems together. It is not mandatory to do things this way - but nicer.

The software will allow each constituent UNIX system to have it's own set of users, user groups and password file and each has it's own super-user. It is possible to unite systems which have already allocated the same numeric uid to different individuals. Each system administrator controls remote as well as local usage, but relies on the remote machine to authenticate users.

The software which implements the connection is a transparent layer which sits between the usual system call interface and the kernel. The software intercepts all system calls, and diverts those which relate to some other system to a server process and then to the Newcastle Connection on the other machine. The results of the system call are passed back to the applications

program, completely hiding whether or not they come from the local kernel.

The tasks of the connection layer are:

Mapping local names from remote systems. For instance, it maps names of open files and devices, processes, internal user id's etc.

Diverting accesses that are in fact to be dealt with by another system using a remote procedure call mechanism.

Providing server processes to carry out requests from a remote machine.

Maintaining relevant parts of the overall system directory structure.

Providing surrogate environments for calls from a remote machine.

There is no requirement to alter the kernel, it is only necessary to recompile user processes with a new library.

Currently, the system is in regular use at Newcastle on 5 PDP11's inter-connected by a Cambridge ring. The software has been provided to one other university in the UK for trials. It is subject to confidential discussions re. commercial exploitation with 6 UK companies. It will eventually be available to UK universities.

There are two extensions which are under investigation. UUL (Unix United Ltd.) provides multi level security on the file systems. It uses encryption to enforce the security barriers and to control re-classification. The prototype of this system has been demonstrated to the MoD. The other extension is NMP (Newcastle Modular redundancy) which uses file/process replication and majority voting to mask hardware faults. The applications programs are unchanged, but run on several machines with hidden voting. The prototype is operational using 3 PDP11's.

The Newcastle Connection technique should be applicable to other systems. However, UNIX seems to have been designed for it; important factors are: the hierarchical file system with moveable current working directory's and root directory pointers; the unification of files, commands and I/O devices into one name space; the ability of user processes to initiate asynchronous activity; the clean kernel interface; and the fact that system call parameters are passed by value. There seems to be no obvious technical obstacle to the idea of connecting all UNIX systems in the country into one °UNIX United' system.

-- END --

This was the best meeting for some years, the atmosphere was friendly, there was lots of good natured heckling from the audience and the talks were all interesting. Dave Tilbrook was the winner of the "Jim McKie - sorry the Newsletter's late" sound-alike competition and Jim McKie was the winner of the lumberjack look-alike competition. Thanks should go to all the Speakers and to Chris Miller and his collegues for organising the Leeds end of the meeting. Also, thanks should go to Cornelia Boldyreff and Emrys Jones for various pieces of organisation, including getting the booking forms out and arranging some of the speakers.

**Design and Structure of an Open Distributed Operating System**

Andrew S. Tanenbaum
Sape J. Mullender

Dept. of Mathematics and Computer Science
Vrije Universiteit
Amsterdam, The Netherlands

The Computer Science Group at the Vrije Universiteit is currently doing research in a number of areas related to networks, distributed operating systems, distributed data bases, portable compilers, and related areas. This talk was mainly concerned with two topics: a small, well-structured rewrite of UNIX (called MONIX) for real-time work and personal computers (with or without networking) and a full-blown distributed operating system called Amoeba.

Both systems are built on top of a small kernel that does the barest essentials of processor management, memory management, and device handling. The kernel implements "tasks" and "processes." A task is a short routine, typically scheduled by an interrupt, that runs to completion. All device drivers run as tasks. Since tasks run to completion, many race conditions caused by inopportune interrupts are eliminated. When an interrupt occurs, all that happens is that a task is queued for execution when the current task is finished. Tasks run in kernel mode.

Processes, in contrast, run in user mode, and may be time-sliced and pre-empted by tasks. The kernel manages full multiprogramming of processes.

The MONIX system primarily consists of a user process that carries out all the V7 file system calls (open, creat, read, write, lseek, mount, chdir, chroot, link, etc.). To make a system call, a user process sends the file system a message via the kernel's interprocess communication facility. The file system is compact (10K bytes), well-structured (no gotos, no locking, no global variables, etc.) and also has some facilities for real-time work (e.g., semi-contiguous files.)

The Amoeba system is also built on top of the kernel. It is based on processes sending and getting messages in a protected way. To receive messages, a process, (say, a file server), picks a large random number, get-port, computes put-port = F(get-port), and then does a GET on get-port. The function F(x) is a one-way function, that is, given x, it is easy to find F(x), but given F(x), it is impossible to find x. Clients who want to communicate with the server send messages to put-port, not get-port, which the server keeps secret. The kernel (or a hardware interface) that performs the GET(get-port) call only accepts packets destined for put-port. Since intruders do not know get-port, they cannot intercept messages.

This port mechanism forms the basis for a distributed operating

system based on capabilities. Each object has a capability consisting of the put-port of the server that manages the object, an object number, a rights field, and a random number that authenticates the object. When a message bearing a capability arrives at a server, the server uses the object number as an index into its tables to locate information about the object. A UNIX-like file server would normally use the i-node number as the object number. When an object (e.g. a file) is created, a random number is generated and stored in the i-node. When a capability for the object arrives, the random number in the i-node is compared to the one in the capability. If they agree, the capability is assumed valid, and the requested operation is carried out.

Alternatively, the random number can be used as an encryption key to encrypt (part of) the capability. If this method is used, the capability may contain rights, such as READ, WRITE, etc. When a capability arrives, the object number field (in plaintext) is used to locate the i-node, which in turn provides the decryption key. When this system is used, the random number field in the plaintext capability contains all zeroes. If decryption yields a plaintext capability with all zeroes in the random number field, the capability is accepted as valid, and the rights field is accepted.

An unusual feature of the file system is the separation of the block server and the file server. The block server allows its clients to read and write raw disk blocks, with no file structure. The file server then turns these into files. This arrangement allows multiple (potentially incompatible) file servers to co-exist in the network, sharing disk space on the same disk.

Yet another part of the file system is the directory server, which maps ASCII names onto capabilities. Given an ASCII name or path, the directory server returns the corresponding capability. The directory server need not know what kind of object the capability is for.

A more sophisticated multi-version file server, which allows concurrent updating in a structured way is also in the works.

Little has been published on Amoeba so far, but a slightly obsolete overview of the whole system can be found in Operating Systems Review, July 1981.

A related project being worked on at the Vrije Universiteit is a cross compiler that consists of front ends for various languages (C, Pascal, etc.). Each front end produces a common intermediate code that is then optimized and fed into a table driven back end. The back end produces target code for the machine described in its input table. Tables are being made for the PDP-11, VAX, 68000, Z8000, 8086, Z80, 8080, and other machines. The whole system runs on UNIX (did you expect RSX-11?).

Sooner or later, all of this software will be release to universities for a nominal fee.

# ALICE

## An Interactive Application Programming Language

## for

## UNIX *

## PRODUCT_DESCRIPTION

Systems Designers Limited,
Systems House,
1, Pembroke Broadway,
Camberley,
Surrey, England
GU15 3XH

Tel: Camberley (0276) 62244
Telex: 858260 SYSDES G

* UNIX is a trademark of Bell Telephone Laboratories

ALICE - PRODUCT DESCRIPTION

## 1.0 Introduction

Alice is a software tool running under UNIX. It provides facilities for developing interactive application packages with a simple and consistent user interface. Its primary use is for applications for the novice or casual user, that can be easily extended or modified in parallel with the user's needs and experience. It therefore adds to UNIX a powerful facility to generate management systems, special user systems, or general office systems, all of which can be used by personnel with little or no UNIX experience. Furthermore it is and has been used to build applications for the knowledgable UNIX user to handle a large variety of problems involving data bases, complicated procedures that require user interaction (e.g., creating new users) or as front ends to sets of related applications.

### 1.1 The Need for Alice

The major benefits of UNIX lie in its simplicity, its generality, and its close matching of facilities to the needs of software development personnel. This means that systems can be both developed and maintained at minimum cost and in the lowest practical timescale, using, whenever possible, software tools and packages already available within the system. Furthermore, these tools and packages have wide application to the needs of users other than the software developer. However, the major drawback of UNIX for the non-specialist, is that the orientation of the interface is to the specialist, and is consequently terse and assumes experience and understanding of the tools and their interaction.

Alice overcomes this problem by providing a user-oriented interface that can serve as a front-end to the basic UNIX tools and facilities.

## 2.0 The User Interface

In a typical Alice application program, the user is prompted for a command line. The prompt will consist of a string, set by the programmer, inside a pair of square brackets as in:

    [Select a command]

The user response to such a prompt is to enter one of the following:

1) An empty line
   The default command is invoked. Usually this is to display a tableau (menu) of the commands.

2) '?'
   A brief explanation of the prompt is output.

3) '??'
   A list of the valid responses to the prompt is given.

4) '?' followed by a command name
   An explanation of the named command is output.

5) 'x' followed by optional arguments
   The 'x' (for Xplain) command invokes a program that displays information about some aspect of Alice programming and/or use.

An 'x' by itself explains the 'x' command.

6) '!' followed by a UNIX command
The UNIX command is interpreted.

7) 'q'
The current Alice application program exits.

8) A command name and optional arguments
The named command is fetched from the Alice data base and interpreted.

When the user enters a command name, in response to a "[]" prompt, the item in the data base associated with that name is retrieved and interpreted.

This interpretation may prompt the user for more information using a question suffixed by one of the following strings:

1) ':'
A value is required.

2) (y/n)
A "yes" or "no" answer is required, with "yes" as the default (i.e., "yes" is assumed if an empty line is given as the response).

3) (n/y)
A "yes" or "no" answer is required, with "no" as the default.

The responses to these prompts may be: an empty line (the default value is used); "?" (an explanation of the question is output); "??" (a list of the valid response is output); the terminal's interrupt character (the current item is terminated); or a valid value (Alice resumes execution).

The only other user interaction is a "<pause>" prompt, which is used to suspend Alice output until the user enters a newline, or any interaction invoked by a program invoked by Alice (e.g., an editor invoked by the item to create a file).

The above describes the total user interaction of Alice and the user (excluding error handling and special cases).

One of the advantages of Alice is the consistency and simplicity of its user interaction. Past experience seems to indicate that a novice user can learn to use interact with Alice programs in about 10 minutes, by using the computer assisted instruction package for Alice, which itself is an Alice application. Furthermore, once the user understands this interaction, the use of new Alice applications requires little or no additional training.

## 3.0 The Development System

The primary inputs to an Alice application are: the user commands read from the terminal, and an Alice data base. To execute a user command, Alice retrieves the item named in the command from the data base, and interprets that item.

Each item of an Alice data base file can hold the following records of information:

1) A short description
This string is displayed in the tableau.

2) A status (on or off)
This element specifies whether the user may select the item for execution. The status "off" is used to prevent an item being selected by the user, for example, when the item is only to be used as a subroutine by other items.

3) A group code
Items may be grouped into subsets that are used to select sets of items for display in a tableau or for a change in status.

4) A list of aliases
The names by which the item may be selected by the user or other items.

5) A monitoring string
The name used for logging usage of this item (similar to command monitoring in the Unix shell).

6) An explanation string
The string output to the user when he/she asks for an explanation using the "?" followed by the item name as a response to the "[]" prompt.

7) A set of command records
These records are interpreted when the item is executed. They consist of guards (conditionals used to suppress interpretation of the record), a keyword, and its arguments. Section 6 provides a list of all currently implemented keywords.

8) Askuser description records.
The command record keyword "Askuser" is used to prompt the user for a value or a "yes" or "no" answer. The arguments to this keyword is the label of a "Askuser" record and the name of the variable to which the response is assigned. The "Askuser" record contains: the record label, the prompt type the default or yes/no values and the question prompt and explanation strings.

9) MASCOT Form records
Alice uses MASCOT Channels to communicate with other processes. The command keyword "Form" can be used to initialize other processes that are attached to an Alice program via message channels. The Form records contain the description of the new processes. (Only available for ANGUS users.)

The source for these Alice items are held in standard TIPs format text files using the Alice profile as briefly outlined above. TIPs format text files are standard UNIX text files using an "*" followed by a record "tag" to separate and name records. Thus any standard UNIX tools can be used to create Alice source files. A forms editor that understands TIPs format is available to facilitate TIPs file editing.

Internally, Alice uses tables of item status, group, name, and alias records. To minimise the effort of creating these tables, a tool,

"Mkmenu" is provided which creates a copy of the standard Alice source with the tables already built and appended to the output file. Alice can accept input in either this object form or the straight text form. In the latter case Alice invokes Mkmenu to create a temporary object file which is then loaded.

One of the intentions of Alice was to facilitate quick and easy development and use. The elapsed time from program creation to program use should be seconds, not minutes. As a result, Mkmenu does not check for syntactic or semantic errors, that don't relate to the actual TIPs format. For the most part this is acceptable. Alice scripts are usually small and the language is simple, thus serious errors are uncommon. However they are not impossible, particularly in scripts that make extensive use of arithmetic expressions, and/or control flow constructs (e.g., Loops, Breaks, Jumptos).

To facilitate "compile" time checking the program "Chkmenu" is used. Chkmenu reads an Alice script and checks each item for a variety of potential errors and/or oversights (e.g., missing explanations or aliases). It checks all expressions for syntax, all arguments for correct use and formats, and where possible for valid values.

The Alice interpreter itself supports a number of debugging aids, that are invoked using supplementary commands at the "[]" prompt level. These supplementary commands allow the programmer/user to:

- Display and/or change the values of the string, number or control variables;
- Examine the code for an item;
- Examine the contents of any message IDAs (MASCOT Interprocess Data communication Areas);
- Edit the source text and reload it using your favourite editor.

## 4.0 The Alice Package

The SDL Alice product consists of the following components:

1) The Alice interpreter

2) Mkmenu
   Converts Alice source text to an Alice object format.

3) Chkmenu
   Offline Alice script syntax and semantic checker.

4) Xalice
   An online documentation package used by both programmer and user to explain Alice interaction, error messages, and keywords. Xalice is used to support the '??' responses to a query and the 'x' command.

5) Programmer Manual sections for the above programs.

6) An Alice programming tutorial.

7) Example Alice scripts including: setting up a new user; a computer assisted instruction package for Alice users; a simple office system; a game parlour including an adventure type game.

If combined with SDL's ANGUS product line (the UNIX/MASCOT kernel and MASCOT development tools) the following extensions are provided:

1) Alice message channel library
Used to build applications that talk to Alice scripts.

2) Example Alice scripts using IDAs.
Computer conferencing script.

3) Alice IDA terminal interface

All other required tools are provided as part of the ANGUS package.

## 5.0 Summary of Alice Benefits

- User acceptance. The simple user interface, which can be used over a variety of applications, reduces the cost of user training and the acceptability of Unix based systems to users without Unix expertise.

- Flexibility. The user interface for the same system, can be changed to match the "knowledge" or requirements of different users without change to the internal working of the system.

- Reduced development overheads and timescales. All the standard Unix facilities are available, as well as the special Alice facilities, aimed at minimising the time, and the cost, of developing new systems.

## 6.0 List of Alice Command Keywords

Anymsg : Get IDA message and assign to string variable.
Append : Append string to a file.
Assign : Assign value to string or number variable.
Askuser : Call subroutine to prompt user for value.
Attach : Attach to IDA message channel file.
Break : Break out of nested command blocks.
Call : Call an item.
Case : Skip to labelled Case.
Cat : Dump named file to output.
Chdir : Change working directory.
Control : Set named control.
Create : Attempts to open file for writing.
Date : Set variable to formatted current time.
Drop : Set variable to string with 1st word dropped.
Echo : Output A-arg and B-arg.
Enable : Enable or Disable list of items.
Exit : Exit from alice with status A-arg.
Form : Form a child process with attaching IDAs.
Ftime : Set variable to a formatted file's modification time.
Getmsg : Get a message from an IDA and assign to variable.
Isnum : Test if string is numeric.
Itemnum : Set number variable to index of named item.
Jumpto : Jumpto a new item or out to top level.
Logfile : Open and/or close logging file.
Loop : Loop back to start of nested command block.

| | | |
|---|---|---|
| Menu | : | Spawn new Alice process with new menu. |
| Newmenu | : | Load new menufile. |
| Pause | : | Pause for carriage return. |
| Perror | : | Output system error. |
| Popword | : | Drop a word from one variable and assign to another. |
| Putmsg | : | Put a message to an IDA. |
| Read | : | Set variable to contents of file. |
| Redo | : | Redo current item from the top. |
| Remove | : | Remove words from a string variable. |
| Reset | : | Reset globals, control, statii. |
| Return | : | Return from the current Item. |
| Setargs | : | Set the special arguments to words of given string. |
| Setitem | : | Name special item. |
| Shell | : | Execute a Shell command. |
| Shift | : | Shift $N arguments. |
| Space | : | Output blank line. |
| Strcmp | : | Compare two strings. |
| Tableau | : | Output tableau of items. |
| Take | : | Set string variable to first word of a list. |
| Test | : | Test a file's attributes. |
| Write | : | Write a string into a file. |

# SOME BENCHMARKS

Andy Tanenbaum
Vrije Universiteit
Amsterdam, The Netherlands

Teus Hagen
Mathematical Centre
Amsterdam, The Netherlands

At the EUUG meeting in Leeds, several vendors displayed their respective wares in close proximity. To take advantage of this opportunity, we wrote two programs and then measured how long it took to compile each one and how long to run each one. Program #1 is a CPU test; program #2 is an I/O test. Program #1 was tested five times, with the type 'word' declared in five different ways, as shown below. The numbers for test #1 are User time in seconds; the times for test #2 and the compile times are Real times. All tests were made with only a single user. The PDP-11, VAX and PERQ tests were made later. The programs follow (PERQ version was in Pascal, the rest were in C).

```
/* Test 1 */
typedef ... word ;
main()
ç word i,j,k;
  for (i=0; i`1000; i++)
    for (j=0; j`10000; j++)
      k = i + j + 1982;
¾
```

```
/* Test 2 */
main()
ç int i, n;
  char a[512];
  if ((n=creat("foo",0755))`0)perror("bah");
  for (i=0; i`500; i++)
    write(n, a, 512);
¾
```

| Test | VAX 11/780 | PDP-11/44 | Zilog | Bleas-dale | Codata | CRDS | PE3210 | PERQ |
|---|---|---|---|---|---|---|---|---|
| #1 register short | 86 | 57 | 73 | 156 | 105 | 158 | 67 | |
| #1 register long | 47 | 218 | 133 | 562 | 105 | 158 | 67 | |
| #1 short | 88 | 112 | 147 | 333 | 171 | 282 | 167 | |
| #1 int | 67 | 112 | 147 | 333 | 193 | 317 | 159 | 182 |
| #1 long | 67 | 218 | 257 | 562 | 193 | 317 | 159 | |
| # 2 | 2 | 32 | 8 | 16 | 28 | 14 | 3 | |
| Compile time test 1 | 3 | 16 | 20 | 20 | 28 | 28 | 4 | 20 |
| Compile time test 2 | 3 | 16 | 20 | 21 | 30 | 42 | 4 | |

Notes on the machines:

| | |
|---|---|
| VAX-11/780: | 2 67MB RM03 drives and 1 256MB CDC 9766 storage module. |
| PDP-11/44: | 2 160MB Ampex Storage Modules |
| Zilog: | 6 MHz Zilog Z8001 with 24 MB Winchester (Zilog Inc.'s system) |
| Bleasdale: | 4 MHz Zilog Z8001 on Multibus with XENIX (Bleasdale BDC 600) |
| Codata: | 8 MHz 68000, memory accessed directly, not via Multibus |
| CRDS: | 8 MHz 68000, memory accessed via Versabus (Charles River Data Sys.) |
| PE3210: | Perkin-Elmer 3210 |
| PERQ: | ICL PERQ (measured at QMC) |

Conclusions: None. Eight isolated tests are not worth much. The purpose of this exercise is to stimulate other people to make more tests. Like in physics and biology, experimental results should only be taken seriously when they can be reproduced by people from different laboratories.

# UNIX Software Catalogue For EUUG members

At the EUUG conference in September I volunteered to carry out the task of building and maintaining a 'software catalogue' of all software publicly available for UNIX. The idea is to have a centralised place where everyone can write to or phone and enquire about what software is available and where from. The benefits are obvious - although a large amount of software is produced at universities, it is very difficult for other universities to find out about this. We desperately need to build up and maintain a list of software.

Although this kind of scheme has been tried in the past, it has not been very successful, mainly due to a lack of response from the EUUG members. For some reason the members don't seem to want to advertise locally produced software. The sort of reasons that were put forward at a discussion at the last EUUG meeting, were as follows :

- no-one wants to give away software they have worked hard to develop.

- no-one wants to spend time distributing tapes containing their software to others.

- no-one wants to spend most of their time answering queries from outsiders about their software, perhaps asking for bug-fixes or how to get the software up and working on their particular system.

All of these points are understandable. However there has to be a little give and take of software. If no-one were to give any software to other members of the EUUG, then there would be no prospects for interchange of software between the EUUG members, which can't be a good thing for any of us.

As to the problem of software distributions, another member of the EUUG has volunteered to distribute tapes if the software is sent to him. However he doesn't want to have to send little bits here and there, so our first task is to build up a tapeful of software arising from different contributors, and keep it centrally. Then a whole copy of this tape will be sent to anyone who asks for it, at a nominal charge.

As to the problem of being pestered by outsiders asking questions about your software, we can make it a condition that the receivers of the tape accept it "as is" with no support. This isn't nearly as bad as it sounds at first; after all one of the biggest distributors of UNIX software (let's name no names) has a very similar policy.

So, before we can start on collecting software and building a tape of it, we need to build a software catalogue, and the sooner we do this the better it is for all of us. Here's what to do. Make up a list of any new pieces of software on your system that you would like to contribute to the software catalogue. The list should be headed something like this :

(1) Title of software
(2) Implementation machine
(3) Description of software
(4) Implementation language
(5) Documentation availability
(6) Outstanding bugs known
(7) Machine dependancies
(8) Original source

Here is a description of what each category should contain; I hope that the motivation behind each is obvious.

(1) Title of software
 - e.g. Graphics package, rewrite of chmod command, Z8000 cross assembler etc.
(2) Implementation machine
 - what machine was this software developed on ?
(3) Description of software
 - a brief description of the application
(4) Implementation language
 - what language is it written in ?
(5) Documentation availability
 - is there any documentation available on its use?
(6) Outstanding bugs known
 - are there any major bugs or restrictions that you know of?
(7) Machine dependancies
 - is this software designed to run only on particular machines? does it require any non-standard features?
(8) Original source
 - if you got it from elsewhere, where did you get it from?

Try and give as much information about the software as you can, but don't worry if you don't have all the info - someone else in the group might be able to work it out. Try and consider what you would like to know about someone else's software, and try and provide such information about your own software. After having made your list, please send it to me at the following address :

Mr. Bipin Dattani,
Perkin Elmer Data Systems Ltd.,
227 Bath Road,
Slough,
Berkshire SL1 4AX,
England

Should you need to contact me by phone, the number is (Slough) 34511

Thank you,

B. Dattani

Bipin Dattani.
Sept 1982

# An implementation of Henderson's SECD machine under UNIX

Mark Dawson

SWURCC

## 1. Introduction

The SECD machine implemented exactly corresponds to the one described in Henderson's book: 'Functional Programming - Application and Implementation'. The effort was primarily an exercise in implementation and took the author about 5 days in all.

An SECD machine is a simple computer, it has 4 registers, the Stack, Environment, Control, and Dump. The execution of the computer is described by transitions of the form:

S, E, C, D --> S', E', C', D'

(selected by inspecting what is the CAR of the Control register). Henderson describes 21 basic transitions, and in his book cleverly and naturally works from a lisp-like language through a description of an SECD machine to a compiler for the language. The interpreter described here executes his definition, and successfully compiles the compiler.

My enhancements to the machine are not described.

## 2. Implementation

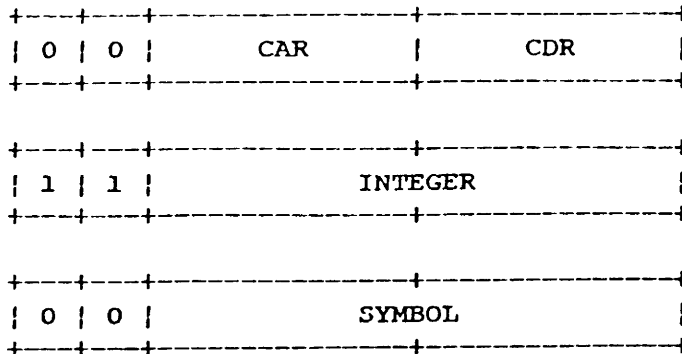
The machine as developed in the book can be split into several parts: S-expression input and output, garbage collection, and instruction execution. The main consideration was the design of the data structures representing the node space, and it is these I describe first.

## 3. Data Structures

The node is essential to the machine, it has three roles: as a integer, as a symbol, and as a pair (with CAR and CDR parts). It is using the node that all s-expressions are constructed, and manipulating s-expressions is the bread-

and-butter of the machine.

The data structures for the node went throught two iterations of design from the original. It is quite interesting to see how they developed from one to the other. Initially I took the simple approach described in Henderson's book, ie:



```
+---+---+---------------+---------------+
| 0 | 0 |      CAR      |      CDR      |
+---+---+---------------+---------------+

+---+---+---------------+---------------+
| 1 | 1 |            INTEGER            |
+---+---+---------------+---------------+

+---+---+---------------+---------------+
| 0 | 0 |            SYMBOL             |
+---+---+---------------+---------------+
```

This is represented by the C data structure (a):

```
        typedef struct node {
                int tag;
                union {
                        long int ival;
                        struct {
                                struct node *car;
                                struct node *cdr;
                        } pval;
                        char *sval;
                } uval;
          } NODE, *PTR;
```

Which is basically a union of the three types; a long integer, a pointer into the symbol space, or a pair (CAR+CDR). The size of this structure came to six bytes. Even using separate I&D spaces I was only able to get an 'array' of 4000 of them - hardly adequate for anything except debugging the machine, certainly not compiling the compiler.

The key to the first development was to realise that the s-expressions used by this machine are mainly linear lists with occasional lists branching to the left. Knowing this it is possible to almost double the storage. The secret is to allow the CAR and CDR parts to be actual values rather than simply pointers to nodes containing the values. In effect creating a psuedo pointer, which sometimes doesn't point anywhere but is itself the value. This leaves the CONS function as the only thing to consume node space - previously calls to 'new_int' and 'new_str' also did.

The following C structures achieve this (b):

```
        typedef struct object {
                char tag;
                union {
                        struct node *ptr;
                        int ival;
                        char *sval;
                } uval;
        } PTR;

        typedef struct node {
                char tag;
                struct object car;
                struct object cdr;
        } NODE, *NODE_PTR;
```

The size of this structure is 10 bytes (instead of 6). Fortunately this gave me enough room to compile the compiler and generally prove the rest of the design but to do anything more required more space.

Fortunately bit stuffing (b) gave the solution. I was able to cut the cost of a node by more than a factor of two, again doubling the amount of freespace available. (Bringing the number of nodes in a separate I&D system to over 12000.)

C allows one to do this relatively cleanly by specifing fields of bits within a machine word and allowing one to manipulate fields as unsigned integers - doing all the fiddling itself. Using this technique the final data structure was achieved (c):

```
        typedef struct node
        {
                unsigned pair: 1;
                unsigned car : 15;
                unsigned tag : 1;
                unsigned cdr : 15;
        } NODE, PTR;
```

Along with each data structure I wrote a corresponding set of C macros. By constraining access to the data structures through these I was able to make the transition from one structure to the next relatively painless, although there was still some unavoidable hacking. Macros do not have the costs in time associated with functions as would be the case with Pascal, say. (Although an good optimising compiler might well expand them in-line...)

The evolution of these macros is shown below.

For (a) above:

```
#define CAR(x) (x->uval.pval.car)
#define CDR(x) (x->uval.pval.cdr)
#define TAG(x) (x->tag)
/* Atom handlers */
#define STR(x) (x->uval.sval)
#define INT(x) (x->uval.ival)
/* Type predicates */
#define ISP(x) ((x->tag&tMASK)==pTYPE)
#define ISI(x) ((x->tag&tMASK)==iTYPE)
#define ISS(x) ((x->tag&tMASK)==sTYPE)
/* Print atoms */
#define PTI(x) printf(" %ld ",x->uval.ival)
#define PTS(x) printf(" %s ",x->uval.sval)
```

For (b) above:

```
#define CAR(x) (x.uval.ptr->car)
#define CDR(x) (x.uval.ptr->cdr)
#define TAG(x) (x.uval.ptr->tag)
/* Atom handlers */
#define STR(x) (x.uval.sval)
#define INT(x) (x.uval.ival)
#define VAL(x) (x.uval.ptr)
#define TYP(x) (x.tag)
/* Type predicates */
#define ISP(x) ((x.tag&tMASK)==pTYPE)
#define ISI(x) ((x.tag&tMASK)==iTYPE)
#define ISS(x) ((x.tag&tMASK)==sTYPE)
/* Print atoms */
#define PTI(x) printf(" %d ",x.uval.ival)
#define PTS(x) printf(" %s ",x.uval.sval)
```

For (c) above:

```
#define CAR(x) (((NODE_PTR)(x<<2))->car)
#define CDR(x) (((NODE_PTR)(x<<2))->cdr)
#define TAG(x) (((NODE_PTR)(x<<2))->tag)
/* Atom handlers */
#define STR(x) (symbols[x&cMASK])
#define lSTR(x,y) (x=sTYPE|y)
#define INT(x) (x&cMASK)
#define lINT(x,y) (x=iTYPE|y)
/* Type predicates */
#define ISP(x) ((x&pMASK)==pTYPE)
#define ISI(x) ((x&tMASK)==iTYPE)
#define ISS(x) ((x&tMASK)==sTYPE)
/* Print atoms */
#define PTI(x) printf(" %d ",x&cMASK)
#define PTS(x) printf(" %s ",symbols[x&cMASK])
/* Type conversions */
#define NTP(x) ((PTR)x)>>2
#define PTN(x) ((NODE_PTR)(x<<2))
```

Notice how the complexity of the macros increases as the data structures become more obscure in form.

## 4. The Freelist

When the interpreter is started one of the first things it does is to grab a large contiguous area of store and to construct a freelist within it. The freelist is a linear list of unused elements, when it becomes empty it is necessary to force a garbage collection to try and reconstruct it.

An interesting idea would be to number the CAR portions of the freelist in such a way that the head of the list indicates the size of the remaining list (this would cost nothing using the 'psuedo' pointer idea). The system could then garbage collect conditionally on the size of the remaining list before certain events.

## 5. S-expression Input and Output

This is an interesting aspect of the interpreter. I used a technique slightly different to the one described by Henderson. Introducing functions for token, token_type, etc, enabled me to simulate lazy input. Unix's input/output is very simple, avoiding the need for line buffers. Any restrictions on line length can be imposed by a filter process tacked on the interpreter by a pipe.

## 6. Garbage Collection

Fundamentally similar to Henderson's, the only change necessary was to improve the performance of the marking procedure. Henderson describes a purely recursive algorithm, but a little thought suggests that the stack requirement can be reduced drastically by making it iterate over the CDR branch. (Making use of the assumptions about the form of s-expressions.)

The main stumbling block for me was an obscure bug in the code of the collection phase. The node before the start of the freelist was also collected onto the freelist (corrupting one marked node). This bug was difficult to find, and only crept in because I didn't test the collector with sufficient care.

## 7. Machine Execution

The SECD machine is represented by a loop which executes instructions repeatedly until a STOP instruction is reached. (The stop instruction is actually defined as the transition which leaves the state of the SECD machine unchanged, ie:

```
S, E, (STOP.C), D --> S, E, (STOP.C), D
```

fortunately the implementation and specification diverge here inorder to allow the machine's state to be displayed!) The evaluation loop has the basic structure:

```
        s = CONS(a,NIL);
        e = NIL;
        c = f;
        d = NIL;

        stop = FALSE;

        do
        {
                i = INT(CAR(c));
                if( debug==TRUE ) show_opcode(i);
                switch( i )
                {

                case pLD:

                . . .

                case pSTOP:
                        stop = TRUE;
                        break;

                default:
                        printf("**ERROR** Unknown opcode0);
                        dump_all( );

                }

        } while( stop!=TRUE );
}
```

Instructions are written very simply by translating their transitions into C. One very important consideration is to avoid nested CONS's - since if a garbage collect occurs during the outermost CONS, there will be no way of marking the inner CONS and it will be added to the freelist. Loosing it forever, viz:

```
     CONS( x,CONS( y,z ) )
```

should be written as:

```
     w = CONS( y,z )
     p = CONS( x,w )
```

The following example illustrates an instruction:

```
     ((c'.e') v.s) e (AP.c) d --> NIL (v.e') c' (s e c.d)
```

```
case pAP:
     w = CONS(CDR(c),d);
     w = CONS(e,w);
     d = CONS(CDR(CDR(s)),w);
     e = CONS(CAR(CDR(s)),CDR(CAR(s)));
     c = CAR(CAR(s));
     s = NIL;
```

The AP (APply function) instruction expects the closure of the function's code (c') with the environment in which the function was declared (e') on the top of the stack. Beneath the closure are the arguments (v) to the function. The previous values of s, e, and c are saved on the dump, d, the stack is NILed, the control becomes the code of the function (c'), and the environment the original environment augmented by the arguments.

**UNIFLEX Evaluation**

Zdravko Podolski

Computing Science Department
University of Glasgow
December, 1981
(A personal view)

**Introduction**

South West Technical Products were kind enough to lend me a Uniflex system for evaluation. The system was similar to that shown at the Nottingham EUUG meeting in September 1981. It consisted of a Motorola M6809 cpu running at 2MHz, 256KB of memory, two 8" floppies, a 20MB Winchester disk and four terminal ports. There were also two SWTP terminals and a serial printer.

Uniflex requires a system with at least 128KB of memory and twin 8" floppies. Such a system would cost about 5000. Expansion is possible up to a total of 1MB memory and several disk drives. It is claimed that the maximum system could support about 20 users.

The evaluation system would cost about 12000, including terminals, printer and software.

**Impressions**

The hardware appears robust, particularly the winchester. I did not like the printed circuit boards in the cpu assembly being supported by their edge connectors. Nevertheless the system performed reliably half an hour after being taken out of a very cold car boot. There were three components to the main system, the cpu box with memory, measuring about a foot by two feet, the floppies in a similar sized box and the winchester in a box about two foot by two foot. The whole lot fits neatly into a desk also supplied by SWTP.

The terminals were SWTP's own. These are extremely powerful intelligent terminals with rudimentary but impressive graphics facilities. Their looks were strange at first sight but one quickly got used to it. The keyboards were high quality and seemingly robust and no fault could be found with the picture quality. However the coating on the screen surface on both seemed to be coming away, giving a curious out of focus image at the corners. These terminals have been carted about and subjected to much abuse so it was amazing that they worked at all. All the terminal parameters could be changed from the keyboard by typing various control character combinations, including the baud rate.

The software too was robust, although sensitive to wild address references. I did not manage to cause appreciable file system damage even though the system crashed several times. The main command interpreter is based on the Unix shell, but with only a small subset of

features. Languages available on the evaluation system were assembler, C, basic and Pascal. I did not have the time to test the Pascal and ignored the assembler. Utilities supplied with the system were only a small subset of those a Unix user is accustomed to, but are increasing in number all the time. One has to pay extra for languages beyond assembler. A very good programmers' editor is available as is a sort/merge package.

**Features**

The file system is closely modelled on the Unix file system. Files may be up to 1 billion bytes long and are considered to be simple collections of characters. I/O devices appear in the file system as special files.

The system calls appear different at the first glance, but from C a Unix like interface is provided. Some incompatibilities still remain, but the suppliers are actively working on removing them. A major addition is a system call enabling one to lock a section of a file (1 to 64K bytes long) for exclusive access. Although simple in concept, this is a viable alternative to the semaphore system as implemented by Glasgow. The locking does not actually preclude writing the 'locked' section, unless the new writer also does a lock system call. This mechanism is thus useful for system programs such as printer spoolers etc, but not general user software. Extending it should not be difficult.

The memory management appears similar to the small PDP11's, with 4KB segment size. A big criticism is that there is no protection against wild addresses, so programs could corrupt one another and the kernel. Similarly, all instructions are available to the programmer and it is possible to mask interrupts or whatever, thus completely disabling the machine. Before the system could be used in a hostile environment such as a student programming system this will have to be fixed. Use in a friendly environment, such as a software house, would be perfectly feasible. The C compiler in particular, generates reliable code.

The system allows programs to be up to 64KB in length, of which at least 4KB is reserved for the stack. Sharing text is allowed. User id's are similar to Unix (8 letters all lower case) and there is no notion of groups.

The shell is a small subset of the Unix shell, allowing pipes, I/O redirection and background processes. Shell scripts are possible, but there does not seem to be any facility for changing the flow of control.

The C language recognised by the compiler is almost that in the 'book'. Exceptions are bit fields and structure assignment. For some unfathomable reason initialisations of variables are not permitted. The compiler is quick and seems to produce good code.

The stdio library is implemented almost in its entirety so most Unix C programs should work on Uniflex. The C compiler and the libraries are undergoing improvement to bring them closer to the standard.

The Basic supplied looks reasonable, if one can say that of any Basic. It would seem to be related to DEC Basic and is similar to the one in Flex. Those used to DEC Basic will have to be careful at first, those converting up from Flex should not have any problems. There are some deficiencies though. The if-then-else is too primitive, not

allowing multiple statements per line after the 'then' if the condition is false. Only simple expression functions are allowed. On the other hand the macro definition facilities look pretty useful.

**Performance**

The performance of what is after all only an 8 bit microcomputer was impressive, particularly in file handling. SWTP claim a swap rate of 180KB per second and a file to file copy rate of about 25KB per second. Some I/O times are given in the following table. They are for copying a 480 block file.

| System | CPU | Elapsed (sec) | msec/block |
|---|---|---|---|
| Unix | PDP11/70 | 21 | 21.8 |
| IAS | PDP11/70 | 19 | 19.8 |
| Uniflex | M6809(1MHz) | 27 | 28.1 |
| Uniflex | M6809(2MHz) | 24.6 | 25.6 |
| VM/UNIX | VAX11/780 | 9 | 9.4 |

The last two figures were obtained here thus justifying the SWTP claims.

**Benchmark timings**

Scripts were run to test the fork/exec performance and the Kashtan benchmarks to test ipc and context switching. The Kashtan benchmarks contain bugs, and the multiprocess versions gave inconsistent timings. These are quoted below, but no attempt can be made on assessing performance. All the tests were repeated three times, except where stated.

**DR1**

Compile and link the ubiquitous C program:

```
main()ç
/* this is a timing test */
long int i;
i = 1;
while ( i!= 10000)
i += 1;
}
```

The times were:

11.3 real, 2.5 user, 4.2 sys
11.3 real, 1.6 user, 4.3 sys
11.8 real, 2.2 user, 3.9 sys

**DR2**

Change into a directory containing 52 files:

```
cd /usr/include ; echo 'ls | wc' | time sh
```

There were 52 files, the names adding up to 783 characters.

```
1.6 real, 0.3 user, 0.8 sys
1.7 real, 0.2 user, 1.2 sys
1.7 real, 0.3 user, 1.2 sys
```

**DR3**

```
cd /usr/include ; echo 'cat * | wc' | time sh
```

There were 3050 lines, 8846 words and about 76000 characters.

```
18.6 real, 10.2 user, 4.3 sys
18.1 real,  9.9 user, 4.8 sys
18.6 real, 10.6 user, 4.3 sys
```

**DR4**

The following was simulated using a file of all the names:

```
cd /usr/include
echo 'for A in * ; do cat $a; done | wc' | time sh
```

What was actually done was:

```
cd /usr/include
echo 'sh temp | wc' | time sh
( where temp was: ' cat f1 ; cat f2 ; ... ' )
```

The numbers were unsurprisingly the same as before, but the times were now:

```
31.8 real, 11.4 user, 13.4 sys
29.9 real, 11.1 user, 12.6 sys
30.2 real,  9.9 user, 15.0 sys
```

From the last two tests it can be calculated that the cost of a fork(), exec(), exit() sequence for the system is:

0.24 seconds real, 0.011 seconds user and 0.18 seconds system time

This is as fast as the fastest 16 bit Unix V7 system. This does not mean that Uniflex will outperform a PDP11 in normal timesharing applications, but that it is very quick in generating new processes.

**Kashtan Benchmarks**

The programs were run, but they contain bugs and also the timings were very inconsistent. Nevertheless the Uniflex system performed the signaling without context switch test at about 60% the speed of the PDP11/45 running fairly standard V7, and the context switching test at 30 to 40%.

The IPC tests show a performance between 20 and 30% of the PDP11/45.

The general feel was of a fairly quick machine, not in the league

of the PDP11/45, but then the cost is an order of magnitude less than the 11/45 was in its day.

**Conclusion**

The Uniflex system is a modestly priced entry level system for people wanting Unix-like facilities. It remains to be seen how the pricing will be affected by the 16 bit microprocessor based competitors. Where the system really comes into its own is when owners of Flex systems (small M6809 based systems) wish to upgrade. A modest investment will buy them a significant improvement in performance and facilities, with a system sufficiently similar to Flex on the surface to avoid alienating the users.

**P.S.**

I have asked Mr. Russell Brown of South West Technical Products to comment on the above and he came up with the following points which I quote:

1) The maximum number of terminals supported on the system is 32. This is with the use of an I/O pre-processor.

2) A new processor card (the MPU-1) is to become available shortly. This card incorporates hardware protection of the memory mapping registers (accessing of which will cause the system to crash), 64 memory maps (the previous board only had one) and improved context switching.

3) The C compiler now has variable initialisation and structure assignments. Bit fields are almost here (being tested as I write).Φ

4) The bug in wc has been corrected (my fault for using an int!!).

And finally I would like to thank SWTP and Russell Brown for their cooperation.

---

Φ April 1982, ZP

Technology Licensing

American Telephone and
Telegraph Company
Guilford Center
P. O. Box 25000
Greensboro, N. C. 27420
Phone (919) 697-5000
Cable: TWX 510 925-1176

Thank you for your recent interest regarding our UNIX* Time-Sharing Operating Systems. We are pleased to announce that UNIX System III can be made available to qualified institutions under an Educational Software Agreement on a per CPU basis for the following rates:

| | |
|---|---|
| Initial CPU | - $800.00 (includes distribution of set of tapes and documentation) |
| Each additional CPU | - $400.00 |
| Each software/documentation distribution | - $400.00 |

Educational institutions may upgrade all CPUs licensed as of September 30, 1982, in their initial request for a UNIX System III license for the $800.00 fee. Your institution should submit one request for upgrading which defines all such licensed CPUs by type, serial number, and location.

Educational institutions may also license UNIX System III under an administrative agreement for the following fees:

| | |
|---|---|
| Initial CPU | - $16,000.00 |
| Each additional CPU | - 5,400.00 |

Full upgrade credit will be given for any prior fees paid for an administrative license on the same machine.

If you feel you qualify for an educational license and wish to apply, please write to:

AT&T
Technology Licensing
P. O. Box 25000
Greensboro, North Carolina 27420

and provide the following information:

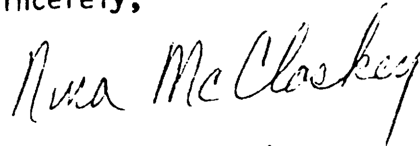
1) The type of software you wish to obtain.
2) The model of computer on which it will run.
3) CPU serial number of that machine.
4) Specific location of machine.

*UNIX is a trademark of Bell Laboratories

5) A statement of your intention to use the software for educational/academic purposes only, as opposed to administrative or commercial use.
6) Type of funding used to acquire the requested software and type of funding for proposed projects that will use the software.
7) Specific academic and educational areas that will use the software (i.e., courses within school curriculum, masters or doctorial thesis, etc.).
8) Authorization by Department Head or the appropriate administrator.
9) Address of your administrative office.
10) Procedures for making the results of research available to the public on a non-preferential basis.

If you find that I can be of further assistance, please do not hesitate to call on (919) 697-5081 or (919) 697-5082.

Sincerely,



Technology Licensing

# UNIX bulletin

*NLUUG*
*Local UNIX-systems Group, Netherlands*

Met dit bulletin willen we een aantal zaken onder uw aandacht brengen. Veel van deze zaken vergen wat spoed, zodat we dit bulletin buiten de EUUG newsletter om naar de leden van de NLUUG sturen. Omdat we tevens wat reklame willen maken voor de NLUUG wordt dit bulletin ook naar andere adressen, waarvan we weten dat men in UNIX† geinteresseerd is, verstuurd. Echter deze adressanten zullen de statuten van de NLUUG niet in dit bulletin aantreffen. Treft u geen statuten aan in uw bulletin en stelt u daar toch prijs op dan verzoeken wij u kontakt op te nemen met het NLUUG-secretariaat: Marten van Gelderen, Nikhef-K, Postbus 4395, 1009 AJ Amsterdam, tel 5922030.

### Kontributie regeling

Vanaf 1 november 1982 wordt de regeling ten aanzien van de kontributie van de EUUG en de NLUUG veranderd. Na die tijd worden alle lidmaatschappen geregeld via de NLUUG. Dat wil zeggen dat de kontributie direkt aan de NLUUG betaald kan worden. Details, zoals de grootte van de kontributie-bedrag, rekeningnummer e.d. vindt u in de NLUUG brochure, bijgevoegd in dit bulletin. In januari 1983 ontvangt u van de NLUUG een faktuur voor het lidmaatschap van de NLUUG en de EUUG. Door uw lidmaatschap van de NLUUG bent u automatisch lid van de EUUG. De nieuwsletter van de EUUG blijft u dus toegezonden krijgen. Ook kunt u gebruik blijven maken van de diensten van de EUUG (networking, software distributies, EUUG meetings etc).
Zijn er klachten (geen newsletter ontvangen, e.d.) dan kunt u kontakt opnemen met het NLUUG secretariaat of via de gemakkelijke weg door het netwerk naar mcvax!nluug.

### 10 december NLUUG meeting

Op **10 december 1982** zal de eerste *officiele* NLUUG meeting plaatsvinden. Plaats: VU, Amsterdam.
Er zijn drie paralelle sessies gepland:

- Lezingen sessie; voorzitter Teus Hagen, MC, Amsterdam.
  Voor deze sessie is iemand uitgenodigd van Bell Laboratories. Wie dat precies is wordt momenteel nog even geheim gehouden. Bovendien krijgt ook u de kans om een lezing te houden. U wordt uitgenodigd om de titel van de lezing en een abstract voor **24 oktober** op te sturen naar Teus Hagen, MC, Kruislaan 413, 1098 SJ Amsterdam.
- UNIX introduktie; voorzitter Hendrik Jan Thomassen, IVV, KU, Nijmegen.
  Deze sessie is bedoeld om de nieuwsgierigheid met betrekking tot UNIX wat te bevredigen. Zij die denken dat UNIX een andere naam voor CP/M is, krijgen de gelegenheid hun kennis bij te schaven. Tevens zullen enige videoopnames van Bell Laboratories over UNIX vertoond worden (voor het eerst in Europa!). Deze sessie is vooral bedoeld voor niet leden.
- Tentoonstelling van UNIX produkten; voorzitter A.S. Tanenbaum, VU, Amsterdam.
  Verkopers van hardware en van software, welke enigzins met UNIX te maken hebben, krijgen de gelegenheid hun produkten tentoon te stellen. Hiervoor is een aparte zaal ter beschikking gesteld. Indien men produkten op 10 december tentoon wilt stellen dan dient men hiervoor

---

†UNIX is a Trademark of Bell Laboratories.

kontakt op te nemen met A.S. Tanenbaum, afd. Informatica, Wis- en Natuurkunde faculteit, de Boelelaan 1081, Amsterdam, tel 5482975.

Gastheer van de meeting is de afdeling Informatica van de Vrije Universiteit te Amsterdam. Vlak voor de meeting zullen nog uitnodigingen verstuurd worden met de details van de meeting.

**EUUG newsletter**

De disk met de adreslabels van de EUUG-computer is drie maanden defekt geweest. En natuurlijk was er geen backup. Dit heeft er toe geleid dat de verzending van de newsletter even op zich heeft laten wachten. Het juli-nummer kunt u nu eerdaags verwachten (tweede week van oktober). Kort hierop kunt u ook het oktober-nummer verwachten. Het oktober-nummer wacht nu slechts nog op de verslagen van de meeting in Leeds. De verwachting is dat volgend jaar de administratieve afhandeling van EUUG zaken sneller zullen verlopen, aangezien vanaf 1 oktober een administratie kantoor in de hand is genomen.

**System III nieuws**

Op de EUUG meeting in Leeds heeft AT&T bekend gemaakt dat de prijzen voor een educatieve license voor UNIX SIII zijn gewijzigd:
Initiële CPU $800, en $400 voor een add-on of een extra kopie van de documentatie en de software. Alle educatieve instituten kunnen hun educatieve license van voor 30 september 1982 eenmalig 'verheffen' tot UNIX SIII voor de totaal prijs van $800.
Een administratieve SIII license voor educatieve instellingen kost $16000, $5400 voor de add-on.
De prijzen voor de andere license-typen zijn niet veranderd.
De aanvragen voor een educatieve license dient voorzien te zijn van het CPU-type, serie nummer van de computer, de lokatie en een aantal gegevens waarmee duidelijk gemaakt moet worden hoe educatief of non-profit men wel niet is (6 items).
Adres: AT&T, Technology Licensing, PO Box 25000, Greensboro, NC 27420,US.

**NLUUG statuten**

In dit bulletin kunt u ook de statuten aantreffen van de NLUUG, zoals die momenteel bij de notaris liggen. Op 10 december zal tevens de algemene ledenvergadering of officiele oprichtingsvergadering gehouden worden. Mocht u aanmerkingen hebben op de statuten, dan verzoeken wij u deze voor 24 oktober aan ons kenbaar te maken.

**NLUUG UNIX brochure**

Omdat er veel vraag is naar UNIX en naar de gebruikersvereniging hebben we een brochure met wat algemene UNIX informatie samengesteld. Aan het bestuur van de EUUG is verzocht om iets dergelijks te doen. Misschien kunnen we de brochures van de EUUG en van de LUUG's bundelen tot een geheel. Bijgevoegde brochure is een aanzet en kan, mits onveranderd, gebruikt worden voor verdere publikatie.

# UNIX (*) for the STD bus

by
Luigi Cerofolini
University of Bologna
(Italy)

(*) UNIX is a trademark of Bell Laboratories.

INTRODUCTION. The STD bus, jontly developped by Mostek and Pro-Log around mid-1978 and now being in the public domain, has gained wide acceptance among designers and, with over 70 manufacturers producing board based products for it, it is one of the fastest growing buses of the past ten years. So it was natural start thinking to make UNIX, one of the most popular operating systems available to-day, running on the STD. The CPU choice was very natural. The STD data bus is 8-bit wide while UNIX was originally designed for a 16-bit CPU: The Intel 8088 CPU and associated co-processors 8089 for I/O and and 8087 for number crunching seemed to satisfy nicely the general system requirements. Because of the small size of the STD boards ( 6.5 in. x 4.5 in.) we have been forced to work on a very modular multi-processors system and this turned out to be the key factor in order to have a very high perfomance system.

We divided the system into four main subsystems or Functional Units (FU): CPU, terminals, disk and tape, memory. In order to offload the CPU from low level peripherals control activities we needed intelligent I/O controllers, but this was contrasting with the very small board size. So we opted for a two-boards set solution for every FU (except memory that fits nicely into one STD board): one of the two boards, the Universal Processor Unit or UPU, is the same for all FUs, while the other bord, the Special Unit or SU, is of course taylored to specific functions. The two boards of the same FU are connected together thru a flat cable.

This possibility of sharing the same hardware between different FUs was very apreciated by all people working for the project: we had the same well known and debugged piece of hardware (the UPU) as the starting point for all new specialized SUs we needed for the system. And this was a decisive fact for the success of the whole project!

THE UPU STRUCTURE. All the UPUs (Universal Processor Units) share the same hardware design and their main components are: the 8088 CPU, one free 40 pins socket (to be used for a cooprocesor like the 8089 or the 8087 or for a Silicon Real Time Operating System Kernel like the 80130) whose usage is very application dependent, latches and STD buffers, logic for DMAing into the sysytem STD bus, local ROM and RAM and a jumpers configuration array.

REMARK. Because of the presence of multiple bus masters, we had to change the (very crude) STD bus contention resolution scheme (BUSRQ*, BUSACK*). We opted for a parallel resolution scheme using a

priority encoder and decoder combination thus permitting up to 8 masters to live on the bus. The XFER logic of the UPU is dedicated to control the data transfer between local and system memory: the local CPU sets the 2-bits XFER_REG before attempting a data transfer.

THE CPU BOARDS SET. In this case the UPU board has the number cruncher 8087 chip into its spare socket and a flat cable takes the highest 8 address bits DA13-DA19 to the associated SU, in this case called the Memory Managment Unit (MMU). The MMU has the responsability to remap these addresses into Physical Addresse PA13-PA19 which are then buffered into the STD. The MMU, in addition to remapping, is also responsible for memory protection and previleged instructions trapping. The MMU module operates in two modes: the system mode, used only by the UNIX operating system, allows modificatipn of the memory map, access of the I/O devices and interrupts. Processes running in user mode are not allowed to use these previleged functions. The MMU unit contains also one programmable timer for system timing requirements (time ticks and timeouts) and one programmable interrupt controller chip.

DISK AND TAPE CONTROLLER. The UPU board has this time one 8089 I/O processor housed into the spare socket. The SU board contains the device registers to communicate with the system, the interrupt logic and an 8-bit general pourpose interface to an intelligent controller-formatter, the SA-1400 from Shugart Corp., for up to four 8" winchester hard disk drivers and to four 1/4" streamer tape drives. The hardware of this SU is very straightforward but, when combined with the intelligence of the UPU, makes the resulting two boards set a very powerful disk and tape controller matching gracefully the UNIX general requirements for such kind of peripherals control.

REMARK: In our implementation of this SU we had a lot of free real estate on the board. So we decided to add two Serial I/O channels, timer and interrupt logic, and one parallel port. This enhanced FU can be used, with the addition of one 256KB memory board, as a minimal (three STD boards set) system to run UNIX (even if with poor memory managment capabilities).

TERMINALS CLUSTER CONTROLLER. In many UNIX systems communication with terminal workstations is usually done in an interrupt-driven I/O one character at a time. Further CPU loading is due to intra-line editing, data flow control, errors and communication protocols handling. Our two boards set for terminals control offloads completely the system CPU from this kind of low level, but very time consuming, tasks. Communication with terminals is handled, as we shall see later, in a very high level and efficient way. The UPU board has the Silicon Real Time Operating System Kernel 80130 chip installed into the spare socket while the SU board has three serial I/O channels, one programmable timer/counter, one programmable interrupt controller, device registers for system communication and interrupt logic.

Also in this case the hardware of the SU structure is very simple,

but, as in the case of the disk and tape controller, becomes very powerfull when combined with the intelligence of the UPU board.

DEVICE HANDLERS. The main CPU and I/O Functional Units (FUs) communicate using a simple but efficient protocol build on a queued command-response structure. When for example CPU has an I/O request for a certain FU it generates an I/O command by adding a new I/O Parameter Block (IOCPB) with an FU tag to the COMMAND_QUEUE associated with the FU. From its side the FU, when ready to process an I/O command points to its COMMAND_QUEUE, process the I/O command and frees the pointed IOCPB retagging it for CPU use (and ,if requested, interrupting the CPU). This buffering of I/O commands permit to the FU also to optimize the processing of commands: for example the disk controller usually tries to optimize disk accesses minimizing heads San Vito dance.

A similar set of operation occurs for the RESPONSE_QUEUE, which also is associated to every FU.

All FUs are very much the same to the operating system and this uniformity has various nice consequences like easy mantainance, short design time, good performance and low cost.

CONCLUSIONS. In order to make UNIX running efficiently on the STD bus we made two basic decisions: the first was the CPU and the associated co-processors choice; the second was multiprocessing with all Functional Units (CPU, Disk and Tape, Terminals) sharing the same intelligence, thus facilitating both the hardware and software system development.

BIBLIOGRAFY

D.Ritchie et alt. The Unix Time-Sharing System, The Bell System Technical Journal, (Special Isssue), July-August 1978, Vol. 57, No. 6, Part. 2.

Intel Corporation. The 8086 Family User's Manual, Oct. 1979.

Pro-Log Corporation. The STD bus. Technical Manual, 1979.

# Prolog - What it is and Where to get it

*Fernando Pereira*

**Why should I want Prolog?**

Prolog is a simple but powerful programming language for symbolic computation based on a computationally treatable subset of logic. It can be seen as a clean combination of the concepts of symbolic programming languages such as Lisp and those of relational databases.

Prolog was born at the University of Marseille in the early 70's. After 8 years of comparative obscurity in a small community of dedicated implementors and users, Prolog has been brought to the attention of the wider world by its surprising adoption as the starting point for the Japanese 5th generation computer research effort.

Prolog has been used for (order of items doesn't imply any form of ranking):

- natural language interaction with computer systems
- architectural design
- drug design (very successful commercial application in Hungary)
- VLSI circuit analysis
- artificial intelligence research
- compiler writing (Prolog itself, APL)
- algebraic computation
- database access and data description languages
- discrete event simulation
- program development systems
- expert systems

and certainly more that I can't remember or don't know about.

**I have a Unix system; how do I get Prolog?**

Please note that Prolog, like other interactive symbolic languages requires more space to run than lower-level languages. Don't expect miracles on a PDP-11.

For the PDP-11 UNIX V6 or V7:

- Chris Mellish's system, obtainable from the Dept. of AI. Edinburgh University, Forest Hill, Edinburgh, Scotland.
- Very compact and reasonably fast, will run substantial programs even without separate I/D space.
- As far as I know, its development has been frozen.
- Written in PDP-11 assembly code.

For the VAX UNIX 4.1 BSD or Eunice under VMS:

- CProlog, obtainable from EdCAAD, 20 Chambers Street, Edinburgh EH1 1JZ, Scotland.
- Designed for machines with 32 bit addresses; requires at least 750K of virtual memory to run comfortably.
- it has an extensive set of system predicates.
- still being developed and improved: if you get the initial licence from EdCAAD, you may ask me for bug fixes and improvements.
- Written in C and Prolog.

Forthcoming:

For the VAX and Z8000:

- POPLOG, combined POP-11 and Prolog from the University of Sussex (available now only for VMS).
- Reported to be somewhat faster than CProlog.
- Written in POP-11 and VAX assembly code.

For 68000:

- EdCAAD's CProlog is being ported.

All these systems comply broadly with the syntax and repertoire of system predicates described in "Programming in Prolog" by Bill Clocksin and Chris Mellish, Springer Verlag 1981. This is the book to get if you want to get into Prolog.

Others:

There are several other more or less portable Prolog systems written in C or Pascal, but they are rather experimental and I cannot recommend them for general use.

If you have information about other Prolog systems, want to know more about Prolog or have bug reports on CProlog, write to:

Fernando Pereira
Artificial Intelligence Center
SRI International
333 Ravenswood Ave.
Menlo Park, California 94025
USA

# C: TOWARD A CONCISE SYNTACTIC DESCRIPTION

Patrick A. Fitzhorn and Gearold R. Johnson
Department of Computer Science
Colorado State University
Ft. Collins, Co. 80523

C is a widely used, low level systems programming language that has found a wide following since its conception at Bell Labs in the middle 1970's. It is the host language of the operating system UNIX, currently available on a wide variety of machines: from the Intel 8086 microprocessor to the venerable IBM System/370. Currently, an effort is even being made at the University of Texas to implement C on a Control Data Corporation Cyber system.

The operating system kernel of UNIX contains 7000 lines of C [John 78], written to be as portable as possible. The C compiler itself operates in two passes and consists of 8000 lines of C, of which about 1500 are target machine dependent in the code generation phase. According to Ritchie, et al [Ritc 78a] C has completely replaced assembly language programming in Unix based systems.

A system recently introduced by BBN Computer Corporation, the C series, is the first machine to be designed with C as its low-level machine language. It also implements, in microcode, the most frequently used constructs of the language allowing for very fast response times.

One of the unique features of C is the structured concept of pointer arithmetic, manipulation and handling as a specific base type. Although some have contended that pointers should not be used [Hoar 75], the type is handled well and quickly becomes indispensable. Pointers to scalar types (integer, character, pointers, etc.) and aggregates (structures, arrays, etc.) are manipulated in a very terse manner. Because of the succinctness of notation however, declarations can become extremely cryptic to generate as well as understand! As an example, consider the following valid function declaration:

```
int (* (* (* Test ()) []) ()) []
```

Test is a function returning a pointer to an array of pointers which point to functions returning pointers to arrays of integers! Once a thorough understanding of pointer manipulation is gained, the notation becomes a help rather than a hindrance, even close to APL in notational conciseness. This feature, along with preprocessing capabilities, structured types built from basic and user definable types (the "enumeration" type documented in 1978), and the extremely rich and powerful set of

operators make C an excellent tool for systems programming.

From the viewpoint of the compiler writer however, the language has severe drawbacks. Anderson [Ande 80] has described C's syntax as "irregular and messy". The cavalier attitude taken by the authors of the language can allow numerous misconceptions of the language's syntactic and semantic capabilities. It would appear that the language's syntax has never experienced a period of rigorous definition and design. This is witnessed in part by the large number of semantics associated with the language. It seems C is given a rough outline in the syntactic description, requiring the semantics to force a viable implementation of the language.

Here then it would seem, a problem exists. On the one hand, C has been targeted for systems programming and implementation, i.e. operating systems and compilers, but due to its vague (and sometimes incorrect) grammar description, the development of a compiler for the language can become a major task.

As an example of the syntactic vagarity, consider the following valid function definition:

```
extern unsigned long int VALID ()  { ... function body ...}
```

Note the long list of qualifiers appearing before the function name, VALID. Extern is a storage class specifier, unsigned long int is the type of the function (a 32 bit unsigned integer). Now note the following function definition in BNF from [Kern 78]:

```
<function definition>  ::= <type specifier> <function declarator> <func-
                           tion body>

<type specifier>       ::= char
                           short
                           int
                           long
                           unsigned
                           float
                           double
                           <struct or union specifier>
                           <typedef name>
```

It is obvious that function VALID cannot be produced from the syntactic description of a function.

Now consider the following invalid function definition:

```
int INVALID () [5]  { ... function body ... }
```

From the definition, INVALID is a function returning an array of five integers. Functions in C, however, may only return a pointer to structures, unions, arrays or other functions. But, from the syntactic

description of a function declarator:

```
<function declarator> ::= <declarator> ( <parameter list> )
                                                          opt

<declarator>          ::= identifier
                          ( <declarator> )
                          * <declarator>
                          <declarator> ()
                          <declarator> [ <constant expression> ]
                                                              opt
```

it is clear that the incorrect function return value for INVALID can be built from the syntactic description. Therefore, it is relatively easy to build constructs that are illegal, or to attempt to build legal constructs that cannot be generated from the syntactic description. It then becomes the purpose of the large number of semantic definitions associated with the language to "weed out" the correct from the incorrect and to allow the correct. It is the purpose of this paper to begin a syntactic description of C that will reduce the number of semantic "hooks" required by incorporating many of the semantic idioms into the syntax, producing a clearer, more complete definition of C.

The syntactic description was specifically scanned in an effort to maximize the correctness of the syntax, while minimizing the number of semantics associated with the grammar. The study focused on three areas of concern in Bell Labs' version of C, whose syntax description is listed in [Kern 78] and [Ritc 78b]:

1. Function definitions and declarations are very incomplete, requiring a large number of semantics to check conditions implementable in the syntax.
2. Statement descriptions are confusing, and in one case in error.
3. Data and function types are used haphazardly in the grammar, with no thought to trap the utilization and scope of storage class specifiers and types.

in addition to the above areas, a study was also conducted concerning SLR parser compatability.

## Function definitions and declarations

One of the annoying discrepencies in C's syntax has already been documented in this paper: illegal function declarations being syntactically allowed, and legal function declarations being excluded. Other areas in the function syntax required close checking also. Three specific areas needing improvement were found:

1. allowing only valid implementations of storage class specifiers and function types, and disallowing any incorrect constructions in function declarations and definitions.
2. requiring functions to return pointers to aggregates (structures and unions), arrays and other functions.
3. separating function declarations from data declarations. Data declarations are extremely broad in syntactic scope, while function declarations are limited.

Storage class specifiers and function typing problems were solved in general by adding a new syntax section called 'type analysis'. This section yields a tightly defined syntactic definition of typing requirements in the language.

According to Ritchie, et al [Ritc 81], a function may not return arrays, structures, unions or functions, although they may return pointers to these types. Bell-C's function declarator, for which the BNF description has already been included, makes no restrictions on these cases. It was felt that the semantics used to inforce these rules could be concisely reproduced in the syntax. Note the following legal and illegal examples:

| legal construct | syntactic description |
|---|---|
| | F is a legal function returning: |
| 1. int F() | an integer |
| 2. char *F() | a pointer to a character |
| 3. union **F() | a pointer to a pointer to a union |
| 4. int (*F()) [] | a pointer to an array of integers |
| 5. struct * (*F()) () | a pointer to a function returning a pointer to a structure |

| illegal construct | syntactic description |
|---|---|
| | W is an illegal function returning: |
| 6. char (W()) [] | a function returning an array of characters |
| 7. int (*W() []) () | a function returning an array of ptrs to functions returning integers |
| 8. char ((*W()) () [] | a function returning a ptr to a function returning an array of characters |

By separating the legal declarations into two parts, a syntactic definition can be constructed allowing legal definitions and declarations while disallowing illegal constructs. The first part allows functions with

no additional function ( '()' ) or array ( '[]' ) qualifiers. This allows declarations of the forms specified by 1. 2 and 3 above. The second part allows a function to return functions and arrays only as pointers to these items. As a byproduct of this division, aggregate types can be neatly excluded from a declaration not returning a pointer.

## Statement description

Consider the following syntax from Bell-C:

```
<statement> ::= <compound statement>
                <expression> ;
                if ( <expression> ) <statement>
                if ( <expression> ) <statement> else <statement>
                while ( <expression> ) <statement>
                do <statement> while ( <expression> );
                for ( <expression> ; <expression> ; <expression> )
                    <statement>
                switch ( <expression> ) <statement>
                case <constant expression> : <statement>
                default : <statement>
                break;
                continue;
                return;
                return <expression> ;
                goto <identifier> ;
                <identifier> : <statement>
                ;

<compound statement> ::= { <declaration list> <statement list> }
                                          opt              opt

<statement list> ::= <statement> | <statement> <statement list>
```

The switch statement in C is analagous to the Pascal Case statement or the PL/1 Select. It evaluates an expression and then selects the proper sequence of code to execute from the constant evaluated. An example from [Kern 78] follows:

```
switch (exp)  {  case '0'  :  /* empty */
                 case '1'  :
                   :
                   :
                 case '9'  :  ndigit [c-'0']++;
                              break;
                 case ' '  :  nwhite++;
                              break;
                 default   :  nother++;
                              break;
              }
```

The switch evaluates the expression 'exp', selecting the proper case to execute. If no case is satisfied, the default (if present) is executed. From the Bell-C description, a case statement is the word 'case' followed by an expression, a colon and a statement. It becomes clear from the syntax that the statement list following case '9', ' ' and the default cannot be constructed. Multiple statements after a colon, according to the syntax, can only be generated using a compound statement, which would generate a semantic error because of the missing left and right braces required for the compound statement.

Also, from the Bell-C syntax, the following illegal statement set can be constructed:

switch ( <expression> ) goto <identifier> ;

This is illegal, although constructable because of the loose syntactic description associated with statements in general. By breaking the statement syntax into statement type descriptors, only correct statements can be generated.

## Data and function typing

This section of the Bell Labs' syntax was the hardest to define and change. A large number of semantic idioms are associated with each syntactic production outlining various allowable type configurations for different declaration contexts. A data declaration can consist of many interrelated constructs which vary with the type (char, struct, etc) and the context (external. i.e. outside of a function; internal, inside a compound statement; and parameter declarations). It is sad to note that most of our time was spent in this section of the grammar, yet it produced the least results. As an example of the variation involved consider the following data declaration from [Kern 78]:

```
extern struct KEY  {  char *KEYWORD;
                      int   KEYCOUNT;
                   }  KEYTAB []        =  {  "break", 0;
                                             "case" . 0;
                                                :
                                                :
                                             "while", 0;
                                          };
```

This declaration defines a structure called KEY which is external. It consists of two members, a pointer to a character (KEYWORD) and an integer (KEYCOUNT). KEYTAB is an array of type KEY and gets initialized with C keywords and an initial count of 0. Note that the array length of KEYTAB is implicit from the initialization. This declaration consists of four distinct parts: a storage class (extern); a data type (struct); a variable declarator (KEYTAB []) and an initialization of the declarator. The pattern is simple, but the construct itself, when semantics are added, is not. As an example, consider a few of the associated semantics:

> The storage class specifier 'register' may only be used with scalar types (int, char or pointer).
> It is not permitted to initialize unions, or aggregates with a storage class specifier of 'auto'.
> When an initialization is applied to a scalar type, it consists of a single expression (a constant if external) possibly in braces.

In the above example, if the initializers were not all present, or not simple constants, the required form would be:

```
. . . }  KEYTAB []   =  {  {"break", 0},
                           {"case" . 0},
                               :
                               :
                           {"while", 0},
                        };
```

The only syntactic elements (in declarations) addressed in this study were storage class specifiers and data typing. By adding two new sections; called 'type analysis', which was discussed above and 'data declarations', discussed below, the problem is partially solved.

The first step taken to provide some syntactic structure was to separate declarations into four distinct groups: external data declarations (outside of a function), internal data declarations (inside a compound statement), function declarations and parameter declarations. Each has its own semantic nuances directly associated with its group classification. Thus, it became an easier task to identify the associated semantics and incorporate them into the syntax if possible. Storage class specifiers were easily trapped, as each group is limited to the type of qualifier that

can be applied to it. The typing problem was partially solved, as was previously stated. The initialization problems were essentially left as is, with minor changes for syntactic readability.

## Bottom-up parser compatibility

The investigation of C as a bottom-up grammar was pursued since the implementation of the language using an LALR parser generator was the basis for this study. Unfortunately, no LALR generator with efficient diagnostics was available. An SLR parser generator was available, so it was used for the tests.

As might be imagined, neither version of C is inherently SLR. Conflicts occur in many areas, with the largest number occurring with the definition of 'primary' and 'lvalue' expressions as follows:

```
<primary> ::= <identifier>
              <string>
              <constant>
              ( <expression> )
              <primary> ( <expression list> )
              <primary> [ <expression> ]
              <lvalue> . <identifier>
              <primary> -> <identifier>

<lvalue>  ::= <identifier>
              <primary> [ <expression> ]
              <lvalue> . <identifier>
              <primary> -> <identifier>
              * <expression>
              ( <lvalue> )
```

An <lvalue> is an expression referring to a manipulatable region of storage, while a <primary> is an expression referring to any usable object. Because the syntactic definition of the two expressions are reasonably close, on SLR item set construction, shift-reduce conflicts occur on overlapping information. Obviously, a new expression class is needed combining these overlaps, allowing greater distinction between the two.

For a comparison of the two grammars, we shall define and use the "conflict ratio". A conflict ratio is simply the ratio between the number of parse table shift-reduce conflicts generated and the total number of item sets constructed. The SLR generator produced 299 item sets for Bell-C with 46 state conflicts for a conflict ratio of approximately 15 percent. CSU-C had 357 states generated. This increase in states was anticipated since many new productions were incorporated in the syntax in leu of semantic hooks. 80 state conflicts occured causing a conflict ratio of 22 percent.

Even with the moderately high conflict ratios, it is anticipated that the grammars could be made SLR with effort. Unfortunately, since an LALR generator was not available, comparable conflict information for that parser could not be gathered. It is anticipated, since LALR handles more general grammars, that the conflict ratio will be lower for both grammars.

## Summary

C is a popular systems programming language for good reason. Its rich set of operators (allowing succinct expressions), pointer manipulation, structured constructs and user definable types all point toward an excellent systems programming language. Unfortunately though, the syntactic description of the language available in the literature [Kern 78], [Ande 80], [Ritc 78b] all leave much to be desired. The only truly objective paper on C in the literature has been Anderson [Ande 80]. He has detailed several errors and weak points in the syntactic description which we have tried to correct and/or implement, along with our own work.

This paper has attempted to define a C that incorporates into the syntax many of the semantics currently describing the language. This should result in easier compiler implementation, and faster compilation. As a justification for an increase in compilation speed consider: instead of executing code associated with certain semantic conditions, a bottom-up parser will be shift-reducing instead, an inherently faster operation.

The authors have not attempted to completely specify C syntactically, but rather have tried to lay the foundation for a structured, correct syntax. We would welcome any comments and constructive criticism of this paper.

## References

[Ande 80] Anderson, Bruce
"Type Syntax in the Language C"
ACM Sigplan Notices 15(3) March 1980

[Hoar 75] Hoare, C. A. R.
"Data Reliability"
ACM Sigplan Notices 10(6) June 1975

[John 78] Johnson, S.C. and Ritchie, D.M.
"Unix Timesharing System: Portability of C programs and the Unix System"
The Bell Systems Tech Journal 57(6) July-Aug 1978

[Kern 78] Kernighan, B.W. and Ritchie, D.M.
"The C Programming Language"
Prentice-Hall, Inc. Englewood Cliffs, NJ 1978

[Ritc 78a] Ritchie, D.M., Johnson, S.C., Lesk, M.E. and Kernighan, B.W.
"Unix Timesharing System: The C Programming Language"
The Bell System Tech Journal 57(6) July-Aug 1978

[Ritc 78b] Ritchie, D.M.
"C Reference Manual"
Interactive System/One Programmers Manual
Interactive Systems Corp Santa Monica, Ca
Oct 1978

[Ritc 81] Ritchie, D.M., Johnson, S.C., Lesk, M.E., Kernighan, B.W.
"The C Programming Language"
The Western Electric Engineer 25(1) Winter 1981

## Appendix

The following is a BNF description of C as developed by the authors.

### *** PROGRAM DEFINITION ***

```
<C PROGRAM> ::= <PROGRAM MODULE>
                <INCLUDE MODULE>

<PROGRAM MODULE> ::= <EXTERNAL DECLARATIONS> <MAIN FUNCTION> <EXTERNAL DEFINITIONS>

<INCLUDE MODULE>  ::= <EXTERNAL DEFINITIONS>


<MAIN FUNCTION> ::= main ( <PARAMETER LIST> ) <PARAMETER DECLARATIONS> <FUNCTION BODY>
                    main ( ) <FUNCTION BODY>

<EXTERNAL DEFINITIONS> ::= <EXTERNAL DEFINITION>
                           <EXTERNAL DEFINITION> <EXTERNAL DEFINITIONS>

<EXTERNAL DEFINITION> ::= <FUNCTION DEFINITION>
                          <EXTERNAL DECLARATION>

<EXTERNAL DECLARATIONS> ::= <EXTERNAL DECLARATION>
                            <EXTERNAL DECLARATION> <EXTERNAL DECLARATIONS>

<EXTERNAL DECLARATION> ::= <FUNCTION DECLARATION>
                           <EXTERNAL DATA DECLARATIONS>
```

### *** FUNCTION DEFINITIONS ***

```
<FUNCTION DEFINITION> ::= <FUNCTION HEADER> <PARAMETER DECLARATIONS> <FUNCTION BODY>

<FUNCTION HEADER> ::= <FUNCTION SC> <FUNCTION TYPE HEADER>

<FUNCTION TYPE HEADER> ::= <SIMPLE TYPE HEADER>
                           <COMPLEX TYPE HEADER>

<SIMPLE TYPE HEADER> ::= <SIMPLE TYPE> <IDENTIFIER> ( <PARAMETER LIST> )
                         <TYPE SPECIFIER> <POINTER> <IDENTIFIER> ( <PARAMETER LIST> )

<COMPLEX TYPE HEADER> ::= <TYPE SPECIFIER> <COMPLEX HEADER> <COMPLEX POSTFIX>

<COMPLEX HEADER> ::= ( <POINTER> <IDENTIFIER> ( <PARAMETER LIST> ) )
                     ( <POINTER> <COMPLEX HEADER> <COMPLEX POSTFIX> )

<PARAMETER LIST> ::= <EMPTY>
                     <IDENTIFIER LIST>

<IDENTIFIER LIST> ::= <IDENTIFIER>
                      <IDENTIFIER> , <IDENTIFIER LIST>

<PARAMETER DECLARATIONS> ::= <EMPTY>
                             <PARAMETER SC> <TYPE SPECIFIER> <PARAMETER DEFINITIONS> ; <PARAMETER DECLARATION
```

```
<PARAMETER DEFINITIONS> ::= <DATA DECLARATOR>
                            <DATA DECLARATOR> , <PARAMETER DEFINITIONS>

<FUNCTION BODY> ::= { <INTERNAL DATA DECLARATIONS> <STATEMENT LIST> }
```

*** FUNCTION DECLARATIONS ***

```
<FUNCTION DECLARATION> ::= <FUNCTION SC> <FUNCTION TYPE DECLARATOR>

<FUNCTION TYPE DECLARATOR> ::= <SIMPLE TYPE DECLARATOR>
                               <COMPLEX TYPE DECLARATOR>

<SIMPLE TYPE DECLARATOR> ::= <SIMPLE TYPE> <IDENTIFIER> ()
                             <TYPE SPECIFIER> <POINTER> <IDENTIFIER> ()

<COMPLEX TYPE DECLARATOR> ::= <TYPE SPECIFIER> <COMPLEX DECLARATOR> <COMPLEX POSTFIX>

<COMPLEX DECLARATOR> ::= ( <POINTER> <IDENTIFIER> ( ) )
                         ( <POINTER> <COMPLEX DECLARATOR> <COMPLEX POSTFIX> )

<POINTER> ::= *
              * <POINTER>
```

*** DATA DECLARATIONS ***

```
<EXTERNAL DATA DECLARATIONS> ::= <EMPTY>
                                 <EXTERNAL SC> <TYPE QUALIFIER> <DATA DECLARATION> ; <EXTERNAL DATA DECLARATIONS>

<INTERNAL DATA DECLARATIONS> ::= <EMPTY>
                                 <INTERNAL SC> <TYPE QUALIFIER> <DATA DECLARATION> ; <INTERNAL DATA DECLARATIONS>

<DATA DECLARATION> ::= <SIMPLE TYPE> <DATA SPECIFIERS>
                       <AGGREGATE TYPE> <DATA SPECIFIERS>
                       <ENUMERATION TYPE> <DATA SPECIFIERS>

<DATA SPECIFIERS> ::= <DATA DECLARATOR> <INITIALIZER>
                      <DATA DECLARATOR> <INITIALIZER> , <DATA SPECIFIERS>

<DATA DECLARATOR> ::= <IDENTIFIER>
                      ( <DATA DECLARATOR> )
                      * <DATADECLARATOR>
                      <DATADECLARATOR> <COMPLEX POSTFIX>

<INITIALIZER> ::= <EMPTY>
                  = <EXPRESSION>
                  = { <INITIALIZER LIST> }

<INITIALIZER LIST> ::= <EMPTY>
                       <EXPRESSION>
                       <INITIALIZER LIST> , <INITIALIZER LIST>
                       { <INITIALIZER LIST> }

<AGGREGATE DECLARATOR> ::= <IDENTIFIER> { <MEMBER LIST> }
                           { <MEMBER LIST> }
                           <IDENTIFIER>
```

```
<MEMBER LIST> ::= <MEMBER>
                  <MEMBER> <MEMBER LIST>

<MEMBER> ::= <INTERNAL SC> <TYPE SPECIFIER> <MEMBER DECLARATOR LIST>

<MEMBER DECLARATOR LIST> ::= <MEMBER DECLARATOR>
                             <MEMBER DECLARATOR> , <MEMBER DECLARATOR LIST>

<MEMBER DECLARATOR> ::= <DATA DECLARATOR>
                        <DATA DECLARATOR> <FIELD>
                        <FIELD>

<FIELD> ::= : <EXPRESSION>
```

*** STATEMENTS ***

```
<STATEMENT LIST> ::= <EMPTY>
                     <STATEMENT> <STATEMENT LIST>

<STATEMENT> ::= <COMPOUND STATEMENT>
                <SWITCH STATEMENT>
                <CONDITIONAL STATEMENT>
                <LOOP STATEMENT>
                <ACTION STATEMENT>
                <NULL STATEMENT>
                <EXPRESSION> ;

<COMPOUND STATEMENT> ::= { <INTERNAL DATA DECLARATIONS> <STATEMENT LIST> }

<SWITCH STATEMENT> ::= switch <EXPRESSION> { <CASE LIST> <DEFAULT> }

<CASE LIST> ::= <CASE STATEMENT>
                <CASE STATEMENT> <CASE LIST>

<CASE STATEMENT> ::= case <EXPRESSION> : <STATEMENT LIST>

<DEFAULT> ::= default : <STATEMENT LIST>
              <EMPTY>

<CONDITIONAL STATEMENT> ::= if ( <EXPRESSION> ) <STATEMENT>
                            if ( <EXPRESSION> ) <STATEMENT> else <STATEMENT>

<LOOP STATEMENT> ::= while ( <EXPRESSION> ) <STATEMENT>
                     do <STATEMENT> while ( <EXPRESSION> ) ;
                     for ( <OPTIONAL EXPRESSION> ; <OPTIONAL EXPRESSION> ; <OPTIONAL EXPRESSION> ) <STATEMENT>

<ACTION STATEMENT> ::= break ;
                       continue ;
                       goto <IDENTIFIER> ;
                       <IDENTIFIER> : <STATEMENT>
                       return ;
                       return <EXPRESSION> ;

<NULL STATEMENT> ::= ;
```

```
*** EXPRESSIONS ***

<OPTIONAL EXPRESSION> ::= <EMPTY>
                          <EXPRESSION>

<EXPRESSION LIST> ::= <EXPRESSION>
                      <EXPRESSION> , <EXPRESSION LIST>

<EXPRESSION> ::= <UNARY EXPRESSION>
                 <BINARY EXPRESSION>
                 <ASSIGNMENT EXPRESSION>
                 <CONDITIONAL EXPRESSION>
                 <PRIMARY EXPRESSION>

<UNARY EXPRESSION> ::= * <EXPRESSION>
                       & <LVALUE>
                       - <EXPRESSION>
                       ! <EXPRESSION>
                       ~ <EXPRESSION>
                       ++ <LVALUE>
                       -- <LVALUE>
                       <LVALUE> ++
                       <LVALUE> --
                       ( <TYPE NAME> ) <EXPRESSION>
                       sizeof <EXPRESSION>
                       sizeof <TYPENAME>

<BINARY EXPRESSION> ::= <EXPRESSION> <BINOP> <EXPRESSION>

<ASSIGNMENT EXPRESSION> ::= <LVALUE> <ASSIGNOP> <EXPRESSION>

<CONDITIONAL EXPRESSION> ::= <EXPRESSION> ? <EXPRESSION> : <EXPRESSION>

<PRIMARY EXPRESSION> ::= <IDENTIFIER>
                         <CONSTANT>
                         <STRING>
                         ( <EXPRESSION> )
                         <LVALUE> . <IDENTIFIER>
                         <PRIMARY EXPRESSION> -> <IDENTIFIER>
                         <IDENTIFIER> ( <EXPRESSION LIST> )
                         <IDENTIFIER> [ <OPTIONAL EXPRESSION> ]

<LVALUE> ::= <IDENTIFIER>
             <LVALUE> . <IDENTIFIER>
             <PRIMARY EXPRESSION> -> <IDENTIFIER>
             * <EXPRESSION>
             ( <LVALUE> )
             <IDENTIFIER> [ <OPTIONAL EXPRESSION> ]
```

```
<BINOP> ::= *
            %
            +
            -
            >>
            <<
            <
            >
            <=
            >=
            ==
            !=
            ||
            &
            &&
            ^

            ,

<ASSIGNOP> ::= =
               +=
               -=
               *=
               /=
               %=
               >>=
               <<=
               &=
               ^=
               |=
```

*** TYPE ANALYSIS ***

```
<TYPE SPECIFIER> ::= <SIMPLE TYPE>
                     <AGGREGATE TYPE>

<SIMPLE TYPE> ::= long <TYPE>
                  short <TYPE>
                  unsigned <TYPE>
                  <TYPE>

<TYPE> ::= <INTEGER TYPE>
           char
           float
           double

<INTEGER TYPE> ::= <EMPTY>
                   int

<AGGREGATE TYPE> ::= struct <AGGREGATE DECLARATOR>
                     union  <AGGREGATE DECLARATOR>
                     <TYPE QUALIFIER>

<ENUMERATION TYPE> ::= enum { <ENUMERATION LIST> }
                       enum <IDENTIFIER> { <ENUMERATION LIST> }
                       enum <IDENTIFIER>

<ENUMERATION LIST> ::= <ENUMERATOR>
                       <ENUMERATOR> , <ENUMERATION LIST>
```

```
<ENUMERATOR> ::= <IDENTIFIER>
                 <IDENTIFIER> = <EXPRESSION>

<TYPE QUALIFIER> ::= typedef <IDENTIFIER>
                     typedef
                     <IDENTIFIER>

<TYPE NAME> ::= <TYPE SPECIFIER> <ABSTRACT DECLARATOR>

<ABSTRACT DECLARATOR> ::= <EMPTY>
                          ( <ABSTRACT DECLARATOR> )
                          * <ABSTRACT DECLARATOR>
                          <ABSTRACT DECLARATOR> <COMPLEX POSTFIX>

<COMPLEX POSTFIX> ::= ( )
                      [ <OPTIONAL EXPRESSION> ]

<FUNCTION SC> ::= extern
                  static
                  <EMPTY>

<PARAMETER SC> ::= register
                   <EMPTY>

<EXTERNAL SC> ::= extern
                  static
                  <EMPTY>

<INTERNAL SC> ::= auto
                  register
                  extern
                  static
                  <EMPTY>
```

Proposed Changes to C

Tim Long
(timl:basservax)

October 23, 1981

Work on C at the Basser Department of Computer Science has spawned some ideas on possible improvements to the language's definition. These are our proposals. We solicit reader's thoughts on this matter.

1. Maxima and minima for arithmetic types

It is proposed that two unary operators be introduced. Their syntax is given by the following addenda to the definition of *expression*. To section 18.1 of *The C Programming Language - Reference Manual* add:

```
maxof expression
maxof ( type-name )
minof expression
minof ( type-name )
```

The syntax is identical to that of *sizeof*, but the type of the argument must be arithmetic. These expression elements are resolved at compile time into the maximum/minimum value attainable by the type of the argument. The type of this value is the same as the type of the argument.

2. Bit stream type

It is proposed that a new data type called *bits* be added, to supercede bit fields. It can also act as a representation for sets. To the type-specifiers in the syntax in section 18.2 of *The C Programming Language Reference Manual* add:

```
bits-specifier
```

with syntax

```
bits-specifier:
     bits { bits-decl-list }
     bits identifier { bits-decl-list }
     bits identifier

bits-decl-list:
     bits-declaration
     bits-declaration bits-decl-list

bits-declaration:
     range ;
     range identifier ;

range:
     constant-expression
     constant-expression .. constant-expression
```

The syntax of the bits-specifier and the bits-decl-list are analogous to the equivalent sections of struct, union and enum declarations. The declaration of a bit stream defines a type which is seen as a stream of at least as many bits as the maximum value found in the bits declaration. For example:

```
bits charset
{
        '\0'      null;
        '0'..'9'  numbers;
        'a'..'z'  lowers;
        'A'..'Z'  uppers;
        maxof(char);
};
...
bits charset   a, *p;
```

Unlike Pascal, bit streams can not be operated on as a unit. There are two ways to reference the components of a bit stream. The first is the extraction into a long or an int (whichever is appropriate) of a named field of a bit stream. This is done in the same manner as struct, union and bitfield member references. For example:

```
a.null          /* will be 1 iff the '\0'th bit of a is set*/
p->numbers      /* will be non-zero iff *p has bits representing*/
          /* numbers set                    */
```

The second is the treatment of a bit stream as an array of bits with reference by an index. For example:

```
a.[ch]          /* will be 1 iff the ch'th bit of a is set*/
p->[ch]
```

## 3. Random initialisation

An additional form of compile time initialisation is proposed to allow the random initialisation of arrays and bit streams. To the initialisers in section 18.2 of The C Programming Language - Reference Manual add:

= set { *random-init-list* }

and to the initialiser-list add:

set { *random-init-list* }

where

```
random-init-list:
     random-initialiser
     random-initialiser , random-init-list

random-initialiser:
     range
     range <- constant-expression
```

a range is described as part of the syntax of a bits-specifier.

A random initialisation must apply to a type which can be indexed, such as an array or bit stream. Such an initialisation will cause all elements in the ranges given to be set to the coresponding constant expression, one if no expression if given. For example:

```
bits { maxof(char); } white_space =
set
{
     ' ', '\n', '\t', '\f',
};

int  a[10] =
set
{
     1..3 <- 5;
     0    <- 1;
     4..9 <- 10;
};
```

# Formatting C

Tim Long

Basser Department of Computer Science
University of Sydney
(timl:basservax)

## 1. Introduction

Every C programmer has strong views on idiom, style and formating. Unfortunately these views are as idiosyncratic as they are inflexible. In C many semantically distinct constructs have only minor syntactic differences. For human beings formatting is often the only reasonable method of distinguishing them.

## 2. Object and type declarations

To establish some terminology we present the following example:

```
static unsigned int     stab_segs, stab_size = 1109, ref_counts[MAX_N];

<----base type---->     <--item-> <--item->          <------item----->

<----first part--->     <---------------second part------------------>

<--------------------------declaration---------------------------------->
```

A declaration usually has two parts. The first part, which we will call the base type, is a list of storage class specifiers, basic type specifiers and adjectival modifiers of basic types. Some examples of storage classes are static and register. The term storage class has lost much of its original intuitive meaning. For instance the modifier typedef is considered a storage class, but it clearly has nothing to do with storage. Examples of basic types are int, float and enum. Examples of adjectives are long and short.

The second part is a comma separated list of items to be declared and their initialisations. Each of these items includes an identifier, possibly surrounded by *, ( ) or [ ]. Any item may be followed by an = and an initial value.

## 2.1. Formatting simple declarations

Only one item should be declared per declaration: there should be no comma separated lists. For example:

```
char    *p, c;          /* WRONG */

char    *p;             /* RIGHT */
char    c;              /* RIGHT */
```

The reasons for this are

(a) all but the first identifier in the WRONG case are hidden and often missed in a quick glance;

(b) the mixture of types (pointer to character and character in the above example) can cause confusion;

(c) it is harder to add a comment or initialisation to an item in the WRONG case.

All base types, items and initialisations within a group of declarations should be verticaly aligned. For example:

```
char *tape_name = "/dev/rht0" /* WRONG */
unsigned long offset; /* WRONG */
int state = st_idle; /* WRONG */

char            *tape_name      = "/dev/rht0";  /* RIGHT */
unsigned long   offset;                         /* RIGHT */
int             state           = st_idle;      /* RIGHT */
```

We can now consider a declaration to have three parts.

(a) The base type, which is never omitted.

(b) The item being declared, which may be omitted.

(c) The initialisation, which will probably be omitted.

It is this three part nature which dominates the layout of simple declarations.

## 2.2. Complex type definitions

The definition of complex types such as structs, unions and enums should be isolated and typedefed. The definition of a complex type in C is a side effect of its appearance in the base type part of a declaration. To make this clearer, consider the following declarations:

```
enum states             state;
struct point            where;
```

Clearly the enum states and the struct point are base types and state

and where are items. Now consider this (badly formatted) example.

```
enum states {st_idle, st_active}        state;
struct point {int x; int y;}            where;
```

This is equivalent to the first example except that definitions are bound to the identifiers states and point. Notice that the definition of the members of the complex type is part of the base type. Finally it should be noted that it is not necessary to bind the complex type definition to an identifer, as the following example shows:

```
enum {st_idle, st_active}       state;
struct {int x; int y}           where;
```

## 2.3. Formatting complex type definitons

The complex type declarations in the previous section were in poor style: a new type name should be created for each complex type generated. There are two ways of doing this. This example demonstrates one:

```
typedef enum                    /* RIGHT */
{
        st_idle,
        st_active,
}
        states;

typedef struct                  /* RIGHT */
{
        int     x;
        int     y;
}
        range;
```

Much of the above formatting will be explained latter. The main point is that the enum and struct are not bound to any identifier. A new type name is created to refer to the types as a whole. The declaration of the objects state and where becomes:

```
states          state;          /* RIGHT */
point           where;          /* RIGHT */
```

Unfortunately this method cannot always be used. When a struct or union references itself (in the form of a pointer) the type of the pointer can not be named because its declaration is not complete. In this situation the following variation can be used.

```
typedef struct struct_node      node;
struct struct_node
{
        int     node_value;
        node    *node_link;
};
```

This binds the definition of the structure to the identifier struct_node in order to achieve a forward reference. But the following declaration is also valid (and preferable):

```
typedef struct node     node;   /* RIGHT */
struct node
{
        int     node_value;
        node    *node_link;
};
```

Notice that this binds the definition of a structure and a new type to two identifiers, both of which are called node. These identifiers come from logically distinct symbol tables. The structure binding is irrevelant and serves only as a mechanism for the forward definition of the type.

Formatting the member list of a complex type is straightforward. The on curly brace should be placed on a new line directly under the base type. The elements of the member list are indented one tab stop, and the formatting rules are applied recursively. The off curly brace is aligned with its matching one. In the second variation this is followed by the semicolon. But if a type name is being defined, the name is placed on a new line indented one tab stop from the off brace, followed by the semicolon.

There are several justifications for this layout.

(a) The conceptually independent acts of type definition and storage allocation are separated.

(b) The indenting and positioning of brackets serves to surround the memberlist declaration with white space, separating it from peripheral activity and placing it where it can be seen and modified. The same arguments apply here as for simple declarations.

(c) The use of a typedef makes the programmer's intention clear.

(d) Subsequent declarations become clean and narrow enough for the author to be consistent with vertical alignment.

The following is a trimmed example of large structure declaration. The source fragments comes from an include file. Near the top of this file is found the following block of typedefs:

```
/*
 *      Forward declarations of general purpose data types.
 */
typedef struct cfrag    cfrag;
typedef struct cnode    cnode;
typedef struct ident    ident;
typedef struct xnode    xnode;
typedef union data      data;
```

Although not all of these forward refrences were necessary all stuctures and unions were given them in this case for consistency.

The following structure definition was found further down the file along with all the other complex type definitions.

```
struct xnode
{
        union
        {
                xnode   *xu_xnd;
                ident   *xu_id;
        }
                x_left;
        union
        {
                xnode   *xu_xnd;
                cnode   *xu_cnd;
        }
                x_right;
        xnode   *x_type;
        xnodes  x_what;
        data    x_value;
        short   x_flags;

};
```

Typical declarations involving this and related types look something like:

```
register xnode  *x;
register ident  *id;
place           where;
```

## 3. Function definitions

```
char    *
strcpy(s1, s2)
char    *s1;
char    *s2;
{
```

The above function definition has a useful characteristic. Although the function returns a non int object, its name appears at the start of a line. This both improves readability and lends itself to automated searching methods. The alternative

```
char    *strcpy(s1, s2)
char    *s1;
char    *s2;
{
```

is readable but does not allow an easy distinction between invocations and the definiton in an editor search. In general the same rules apply to a function definition as a simple type except that a new line is taken immediately before the identifier.

The leading bracket of the formal parameter list should be placed immediately after the function name. The formal parameters themselves should be placed on the same line with a space after each comma. The closing bracket should be placed hard against the last formal parameter (or the opening bracket if there are no formals). For example:

```
main(argc, argv, env)

main( )
```

Declaration of the formal parameters follows, hard against the left margin and obeying the rules of simple declarations.

## 4. Formatting blocks

Blocks have two parts, surrounded by curly braces. These parts are

(a) declarations local to this block;

(b) executable statements.

Where the block is the body of a function the on curly brace is placed on a line of its own, hard against the left margin. Each time a sub-block is opened the on curly brace is indented one further tab stop from the level of the enclosing block. The brace always appears on a line of its own. For example:

```
while (i < n)
{
        dothis( );
        dothat( );
}
```

This positioning of the opening curly bracket is important to

(a) visualy seperate the body of the block from surrounding peripheral acitivity;

(b) act as a pointer to any flow control construct controlling the block;

(c) allow a similar visual clue to any controling expression.

Placing the opening curly brace on the end of the previous line both embedes any controlling expression in blocks of text and leads to special cases when blocks are opened to gain local variables.

The local declarations are started on a new line indented one tab stop from the initial brace. Formatting is as described above. One blank line should be left between the local declarations and the executable statements. If there are no declarations the code should start on a new line immediately after the on brace. For example:

```
{
        char    *p;
        int     i;

        p = "this is a demo";
        {
                i = 0;
                return i;
        }
}
```

The occasional blank line between executable statements is acceptable but should be not be over-indulged. The significance of such blank lines is easily lost. Often a block comment is more appropriate (see "Comments").

5. Formatting executable statements

Statements are placed on new lines indented one tab stop from the level of the on and off braces of their surrounding block. Its is unacceptable to have more than one statement on one line.

```
i = 0; j = 10;                  /* WRONG */
return;                         /* WRONG */

i = 0;                          /* RIGHT */
j = 0;                          /* RIGHT */
return;                         /* RIGHT */
```

Placing many statements on one line banishes all but the first to oblivion. Although it may be argued that some statements are logically related this is not sufficient justification for the devaluation of statements tacked onto the end of another.

## 6. Formatting expressions

When an expression forms a complete statement, it should, like any other statement, occupy one or more lines of its own and be indented to the current level. Binary operators should be surrounded by spaces. Unary operators should be placed hard against their operand.

```
* p ++;                  /* WRONG */
i=i*10+c-'0';            /* WRONG */

*p++;                    /* RIGHT */
i = i * 10 + c - '0';    /* RIGHT */
```

The ternary operators ? and : should also be surrounded by spaces.

When a sub-expression is enclosed in brackets, the first symbol of the sub-expression should be placed hard against the opening bracket. The closing bracket should be placed immediately after the last character of the sub-expression.

```
a = b * ( c - d );       /* WRONG */

a = b * (c - d);         /* RIGHT */
```

Note that the symbols ->, ., and [] which build up primaries (factors) are not considered binary operators in this context. They should not be surrounded by spaces. For example:

```
addr = addrs[ (d >> 3) & 037 ];          /* WRONG */
addr -> csr = 0;                         /* WRONG */

addr = addrs[(d >> 3) & 037];            /* RIGHT */
addr->csr = 0;                           /* RIGHT */
```

The round brackets which surround the arguments of a function call attract no spaces.

```
puts ( "hi\n" );         /* WRONG */

puts("hi\n");            /* RIGHT */
```

Commas, whether used as operators or separators, should be placed hard against the previous symbol and followed by a space.

```
write(2,"whoops\n",7);           /* WRONG */

write(2, "whoops\n", 7);         /* RIGHT */
```

White space in expressions is useful as much by its lack as its presence. For instance placing spaces in the inside edges of brackets merely spreads out the expression and loses the suggestion of binding. Excessive white space causes inflation and promotes devaluation.

Occasionally expressions become too large to fit on a single line. Breaking at an arbitrary column is distasteful and often unreadable. Rewriting the expression as two, possibly using a temporary, may destroy its conceptual integrity and efficiency. The solution is to reformat the expression over several lines. Consider the following:

```
fprintf
(
        stderr,
        "%s: Could not open %s for reading. %s\n",
        my_name,
        tape_name,
        errno > sys_nerr ? "" : sys_errlist[errno]
)
```

This demonstrates the formatting of the most common cause of long lines, the function call with many arguments. Not the position of the opening and closing brackets. The actual para rs are aligned vertically one tab stop in from the current level. I actual parameter occupies a line of its own.

```
if
(
        (id->id_type == NULL)
        ||
        (
                (id->id_type->x_what == xt_arrayof)
                &&
                (item->x_left->x_what == xt_arrayof)
                &&
                (id->id_type->x_subtype == item->x_left->x_subtype)
                &&
                (id->id_type->x_flags & XIS_DIMLESS)
        )
)
```

Here we see another common line length transgressor put in its place. Notice the placement of binary operators and brackets on lines of their own.

The basic message in the above exmaples is don't be afraid of using more lines to make the expression clear.

## 7. Formatting flow control constructs

In order to give visual distinction between flow control constructs (such as for and while) and function calls, a small variation in formatting is introduced. A space is used to seperate a flow control keyword from any controlling expression. For example:

```
if (p != NULL)
{
        dothis( );
        dothat( );
}
return p;
```

The space seperates the keyword in order to emphasise the flow control dominating the following statement or block.

In view of the above the formatting of for and while statements is straightforward:

```
for (p = root; p != NULL; p = p->next)
        process(p->data);

while ((c = getchar( )) != EOF)
        putchar(c);
```

When formatting if statements several alternatives are possible. The simple if statement is again straightforward:

```
if ((fid = open(name, O_READ)) == SYSERROR)
        perror(name);
```

In a simple if-else combination the else keyword should be placed on a line of its own at the same indentation as the if:

```
if (c == '\\')
{
        ...
}
else
{
        ...
}
```

Although these are the only variations of if statements distinguished in the language the author feels that it is often desirable to consider an if-else chain as a flow control construct in its own right. In this case the following layout is acceptable:

```
if (c == '\\')
{
        ...
}
else if (c == '"')
{
        ...
}
else if (c == '\'')
{
        ...
}
else
{
        ...
}
```

The formatting of switch statements is simple:

```
switch (pid = fork( ))
{
        ...
}
```

However the placement of case labels and labels in general often gives trouble. The keyword case should be placed on a line of its own at the same indent level as the controlling switch keyword. A space should separate the word case from the constant expression which is immediately followed by the colon. A blank line should be left above a case label if program flow does not fall through it. For example:

```
switch (pid = fork( ))
{
case SYSERROR:
        fprintf(stderr, "%s: Could not fork.\n", my_name);
        exit(1);

case 0:
        ...
}
```

Ordinary labels and defaults follow the same rules.

Placing executable statements on the same line as a label (of any sort) is unacceptable since

(a) the statement is visualy hidden by the label;

(b) it is impossible to be consistent with indenting, there will always be some constant expression too long.

The formatting of do statements is difficult. The intuitive method

is:

```
	do
	{
		...
	}
	while (...);
```

However the duality of the <u>while</u> keyword often leads to confusion, especially if the preceeding block is large. To avoid this an arbitary convention is adopted (as in the case of flow control keywords and function calls). The <u>while</u> keyword should be indented one tab stop from the level of the closing brace:

```
	do
	{
		...
	}
		while (...);
```

## 8. Comments

Much of this document has concerned itself with formatting aimed at improving readability. The tacit assumption is that readable code is easier to understand than unreadable code. Comments do not improve readability but attempt to directly aid understanding and maintenance.

Comments embedded in code tend to create a dense mass of text. Comments which begin and end on the same line, intermixed with code, should be avoided. It is better to use a few large comments than many smaller ones distributed through the text.

```
	/*
	 *	This demonstates the layout of a "block comment".  One
	 *	comment such as this at the head of a hundred line
	 *	function is often more useful than hundreds of two or
	 *	three worders.
	 */
	main(argc, argv)
	int	argc;
	char	*argv[];
	{
```

```
/*
 *      Block comments such a this and the above should follow
 *      the level of the code they refer to.
 */
if (...)
{
        /*
         *      Indented when the code is indented.
         */
         ...
}
```

One of the most important aspects of comments is their semantic content. Cryptic references should be avoided, "in" jokes should be obviously irrelevant. Comments should contain either

(a) complete english sentences, with capital letters and full stops (periods);

(b) some sort of well defined logical symbolism;

(c) diagrams.

For example:

```
/*
 *      Warning!
 *      i + strlen(str) + base - p <= BUFSIZ
 *      or else.
 */


/*
 * The shape of the file is thus:
 *
 *        -----------------
 *       | header          |
 *       |-----------------|
 *       | hashtable       |
 *       |   (hashsize *   |
 *       |    TABENTLEN)   |
 *       |-----------------|
 *       | table           |
 *       |   (tabsize *    |
 *       |    TABENTLEN)   |
 *       |-----------------|
 *       |  entries        |
 *       \        .        /
 *                .
 *       /        .        \
 *       |        eof      |
 *        -----------------
 *
 * I hope this is a little clearer now.
 */
```

# C Style and Coding Standards

Dennis F. Meyer

Uniq Computer Corporation

ABSTRACT

This paper presents a series of standards for the coding of programs written in the C language. These standards are intended to maximize the readability and maintainability of C programs. No attempt is made to specify or influence the functional organization of a program; this is intentionally left to the discretion and creativity of the programmer. All code authored by Uniq Computer Corporation will adhere to these standards.

January 22, 1982

# CONTENTS

1. Introduction....................................1
   1. Definitions...................................1
   2. Summary.......................................2

2. Files...........................................2
   1. Organization..................................2
   2. Definition Files..............................4
   3. Header Files..................................5
   4. Naming Conventions............................5

3. Variables.......................................6
   1. Declarations..................................6
   2. Initialization................................7
   3. Naming Conventions............................8
   4. Global........................................9

4. Coding Conventions..............................9
   1. Conditional Statements.......................10
   2. Loops........................................10
   3. Compound Statements..........................11
      1. Conditional...............................11
      2. Loops.....................................12

5. Commenting Conventions.........................12
   1. File Description.............................12
   2. Function Description.........................13
   3. Block Comments...............................14
   4. Short Comments...............................15
   5. Appended Comments............................15

6. Miscellaneous..................................15

7. Conclusion.....................................15

# C Style and Coding Standards

Dennis F. Meyer

Uniq Computer Corporation

## 1. Introduction

This document presents a series of standards for the C language style and coding. The intent is to provide a means for maximizing the readability and maintainability of software written in C. Although the functional organization of such software is not addressed, a logical structure and organization is assumed, in keeping with good programming practices; any software with an ill-conceived organization or illogical structure will defeat any attempt at its maintenance.

### 1.1. Definitions

The following definitions are applicable to terms within this document.

application: a collection of loosely related software that performs all the processing required to fulfill the objectives of a project.

global: a symbol that is referenced by more than one object; also called "external".

module: a cohesive set of procedures that perform a distinct data processing operation.

object: a relatively independent entity in a file, such as a set of one or more macro definitions, or a function.

program: a cohesive set of modules, one of which must be a "main", that performs a subset of the processing required to fulfill the objectives of a task.

system: the local operating system (monitor, compilers, utilities, etc.).

task: a series of one or more cooperating programs that performs the processing required to fulfill related objectives of a project.

## 1.2. Summary

Following is a brief description of each of the subsequent sections in this document:

Files: discusses the organization of the files that are used in the production of a program.

Variables: discusses the declaration and documentation of variables used in a program.

Coding Conventions: discusses techniques for visually indicating the structure of a program.

Commenting Conventions: discusses the use of comments in the body of a file.

Miscellaneous: assorted recommendations for coding practices.

## 2. Files

All files applicable only to a given program, task, or application should be maintained in a manner that enables them to be readily identified. File names should, as much as is reasonably possible within the confines of the operating system and/or naming convention restrictions, be indicative of the function(s) performed by the object(s) contained in the file.

Each file is to contain one or more logically related objects. Recall that the term "object" is used here to designate some sort of relatively independent entity, such as a set of one or more macro definitions, or a function.

### 2.1. Organization

A file consists of various sections that are separated by several blank lines. Although there is no maximum length limitation, files should be kept short enough to enable relatively easy editing, compilation, and/or perusal.

Lines within a file, where possible, should be short enough to enable their display without overflow on most terminals (currently 80 columns). Lines filled with asterisks or some other character should be used sparingly, only as an attention-getting mechanism for some particularly complex or critical piece of text.

The order of the sections in a file is as follows:

A. File Description - a comment describing the function and contents of the file (see 5.1); this section is required in all files.

B. "includes" - all header files applicable to any object in the file. Each declaration should be on a line by itself beginning in column one, and should include comments briefly describing the contents of the file and the reason(s) for its inclusion. For example:

```
#include <stdio.h>     /* Define Standard IO macros */
#include "common.h"    /* Define data structures */
                       /* for inter-module
                        * communication
                        */
```

This section is applicable only if files are to be "included". See section 2.3.

C. Definitions - all definitions (e.g., typedefs, defines, structure or union templates) that apply only to this file. Each definition begins on a line by itself in column one, and should include comments describing the defined datum. If any structure templates are defined, each member declaration should include comments describing its use and/or function. This section is optional; it should appear only if applicable.

D. Globals - declarations for global variables applicable only to this file. Each declaration should be on a line by itself beginning in column one, and should include comments briefly describing the declared datum. All variables in this section should be declared "static", in order to explicitly limit their scope; e.g.,

```
static char Buffer ;  /* Temp buffer */
```

This section appears only if applicable.

E. Functions - any function(s) in the file come last. Each function should begin on a new page (insert a control/L into the text), and the body of the function should fit on one page, if possible; only in very rare cases should the body of a function be permitted to exceed two pages. The recommended organization of each function follows:

1. Function Description - a comment that gives the name of the function and a description of what it does. See section 5.2.

2. Function Declaration - the type of value returned by the function should be declared first, on a line by itself, with the "static" keyword prepended if the function has a local scope; functions that return no value should not be given an explicit type.* The function name and formal

---

* If the compiler supports the "void" type for func-

parameters then follow, also beginning in column one. Each parameter should then be declared, each on its own line in column one, together with comments explaining the parameter. The opening brace for the function comes last, again on a line by itself in column one.

3. Local Variables - definitions for all variables referenced only by this function, whether automatic or register. Register variables are particularly beneficial as pointers to structures. Variables that are designated registers should be ordered by priority; i.e., put the variables you want most to be registers first. All definition lines are to be tabbed over by one tab stop. In most cases, each variable should be defined on a line by itself, and includes comments explaining the function and use of the variable. Exceptions to this rule include:

   - exceedingly obvious variables, such as "temp", need not be commented, and may be grouped in one definition.

   - multiple variables with the same function and usage, such as two pointers to the same character array, may be grouped in one definition line.

4. Code - the actual code that performs the function.

## 2.2. Definition Files

Definition files depart from the normal source file organization in that they contain no executable code. Definition files are used to declare and allocate global variables and data structures. They may, in addition, contain typedefs, defines, and structure or union templates that serve to clarify the definitions and/or initial values. Therefore, a definition file can contain only sections A through D of the file organization. Note, however, that the global variables that are to be referenced by actual functions cannot be declared as "static".

Definition files are required for programs and modules that have multiple source files, unless they have no global variables.

---

tions that return no value, such functions should be given that type.

## 2.3. Header Files

Header files differ from the normal source file in that they contain no executable code and no declarations that result in the allocation of memory. Header files are used to provide a means for inserting sets of definitions and/or declarations into other source files. The contents of a header file is inserted into the applicable source by "include" statements in those sources. Since some compilers do not allow nested "includes", and in order to preserve the clarity of a program, header files may not contain any "include" statements. Header files, then, may contain typedefs, defines, structure or union templates, and global variable declarations that do not allocate memory. Therefore, only sections A, C, and a variation of section D may appear in a header file. All global variables mentioned in a header file must be declared as "extern".

Note that the processing of "included" files is dependent on the compiler and the operating system being used. Some compilers may place a limit on the number of files that can be included in a source file. For operating systems that support multiple or hierarchical directories, there may be complications in the manner and sequence in which directories are searched to locate a header file. Some operating systems implement a standard directory search algorithm, which the compiler may or may not choose to use. Some compilers may require that header files reside in a particular directory, or in the "current" directory, or in the same directory as the source file, etc. Others may allow the user to specify or modify the search algorithm at run time. Since there is such a potential for confusion, some convention should be established and documented for each application, so that header files can be located by compilers and by humans.

## 2.4. Naming Conventions

Some sort of comprehensive file naming convention should be established (and documented) for each non-trivial application. Particular conventions will depend on the complexity of the application and on the limitations imposed by the operating system under which the application is developed. Most operating systems will impose their own standardized suffix (or extension) conventions for source, binary, and library files, etc. For example, UNIX* requires C source files to end in ".c", assembler source files in ".s", etc. The user is generally free, however, to standardize suffix (or extension) conventions for header files, although ".h" appears to be a de-facto standard. Conventions for the first part of the file specification are

*UNIX is a Trademark of Bell Laboratories.

limited by the number of characters allowed by the operating system. As a general rule, the following can be used to generate a naming convention for any operating system:

- Allocate the first "n" characters to identify the program
- If the complexity of the application warrants that program modules be split into more than one file, allocate the next "m" characters to identify the module
- All remaining characters are to be used to identify, as much as is reasonably possible, the contents and/or function of the object(s) in the file

For an operating system that supports sub-directories, much of the naming convention can be off-loaded from the file name onto the directory structure. For example, under the UNIX operating system a directory can be established for each application; this directory would contain files common to tasks in the application, and directories for each task in the application. The task directories would contain files common to programs in the task and, if necessary, directories for each program in the task. The program directories would contain files common to modules in the program and, if necessary, sub-directories for each of those modules. Each module directory would contain files unique to that module. This type of structure is recommended, if it is possible, since it will allow for the generation of more meaningful file names.

## 3. Variables

One of the key factors in debugging and/or maintaining a program is an understanding of its use of variables and data structures. This is especially true for global variables, and ultimately true for global data structures. It is essential, then, that all such items be exhaustively documented, by narrative in a separate file, if appropriate. Failure to document such items is to be considered a capital offense.

### 3.1. Declarations

The documentation of variables and data structures in a separate file does not relieve the programmer of the responsibility for commenting such items when they are declared in the source file; it only reduces the amount of explanation required. The general rule for declaring variables is to place one declaration per line (exceptions have been previously noted). The comments which explain the variables may be block comments, short comments, appended comments, or any combination of these, depending on the personal preferences of the author.

All declarations should precede any executable code. Although it is valid C to place declarations within blocks of code, this practice is discouraged, since the potential for confusion is great.

When structures or unions are declared, the opening brace for the member declarations may be placed on a line by itself, immediately below the first character of the declaration, or it may be placed on the same line as the declaration, separated from the declaration by one space. Each member declaration should be placed on a separate line, tabbed to the right by one tab stop. The closing brace should be on a separate line, in the same column as the first character in the declaration. If any variable is declared, it should be on the same line as the closing brace, separated by a single space. For example,

```
struct time         /* Time of day template */
{
     int tim_hour ;           /* 0 - 23 */
     int tim_minute ;         /* 0 - 59 */
     int tim_second ;         /* 0 - 59 */
     int tim_jiffy ;          /* 1/60th of a second */
     long tim_ticks ;         /* Jiffies since midnight */
} start ;           /* Time program started */

struct time end ;             /* Time completed */

struct date {       /* Day of year template */
     int da_day ;             /* Day in month */
     int da_month ;           /* Month in year */
     int da_year ;            /* Year, A.D. */
     int da_inyear ;          /* Day in year */
} creation ;        /* Initial entry date */

struct date update;           /* Date last modified */
```

## 3.2. Initialization

The initialization of variables when they are declared can produce confusing text, especially for multi-dimensional arrays and arrays of structures. For simple variables and short arrays, the initial values may be placed on the same line as the declaration; for other cases, initial values on succeeding lines should be tabbed to the right by one tab stop. The equals sign preceding the initial value should always be included. In general, all initialization text should be made as pretty as possible; comments should be included where appropriate.

## 3.3. Naming Conventions

A desirable feature in any program is to be able to identify a general class of variable easily within the text of the code. For this and other reasons, the following naming conventions should be followed:

- symbols that begin with an underscore are generally reserved by the system software; therefore, unless the object is to become part of the system software, no symbol should begin with an underscore.

- typedefs, macros, and constants should be all upper case and may not contain any underscores; the only exception to this rule is for system software, where a macro may also exist as a function on other systems (e.g., "getchar", "putchar").

- bits and masks that refer to a particular variable or type of variable should be all upper case and must contain an underscore. Characters to the left of the underscore are to be used to identify the variable or type of variable, and are to be the same for all related symbols. Characters to the right of the underscore are to be used to identify the function of the symbol.

- global symbols must have their first character capitalized and all subsequent characters lower case, like a proper name. One-character globals are prohibited.

- global symbols that are referenced only by a particular module should all begin with an "n" character module identifier, optionally followed by an underscore.

- local symbols are to be all lower case, and may contain one or more underscores to increase the clarity of the name.

- structure and union member names should begin with a two or three character structure or union identifier, followed by an underscore.

Within the confines of the above, all symbol names should as much as possible be indicative of the function and/or usage of the variable to which they refer. Variable names that exceed the compiler limit on length are encouraged (to increase the clarity of the name) providing that no compiler rules are violated and that the names are unique within the number of characters recognized by the compiler. Adherence to these conventions will not only increase the clarity of the code, but will produce beneficial side-effects in the generation of cross-reference listings; all symbols of a like class will appear grouped together, enabling easier

analysis of their actual use.

### 3.4. Global

Extreme care should be taken in the assignment and use of global variables, since inconsistent usage is a frequent cause of program bugs, and one of the most difficult to track down.

If a module or a program uses global variables and is of sufficient complexity to warrant multiple source files, all global variables should be defined in separately maintained and compiled definition files.

## 4. Coding Conventions

This section describes conventions that are to be applied to the text of the code in a file. These conventions are intended to force the text to reflect the block structure of the code. As a general rule, the appearance of the text must make this structure visually obvious. All statements within a block of code must be indented by one tab stop from the enclosing block; multiple statements on a line are prohibited. If an expression or statement is too long to fit on one line, it should be broken at a logical point, and the continuation line(s) should be indented from the originating line. The following rules and recommendations pertain to the format of the text within a statement or expression:

- keywords that are followed by expressions in parentheses should be separated from the left parenthesis by one space.

- for function calls and macros with arguments, the left parenthesis should be placed immediately adjacent to the last character of the function or macro name.

- spaces may be placed after commas in argument lists as an aid in visually separating the arguments.

- there should be no space between primary expression operators (i.e., () [] . -> ) and their operands.

- there should be no space between a unary operator and its operand.

- all binary operators should be separated from each of their operands by one space.

- the first expression in the conditional operator should be parenthesized (e.g., max = (a > b) ? a : b).

- the operator symbol in an assignment operator must

immediately precede the equals sign (e.g., +=, *=, etc.)

- expressions containing mixed operators should be fully parenthesized.

- the statement terminator character may be separated from the statement by one space.

The following sub-sections describe conventions for complex statements.

## 4.1. Conditional Statements

Conditional statements contain statements that may be executed based on the results of the evaluation of one or more expressions. The statements that are conditionally executed should be placed on a separate line and indented by one tab stop.

```
if (expression)
     statement ;

if (expression)
     statement ;
else
     statement ;

if (expression)
     statement ;
else if (expression)
     statement ;
else
     statement ;
```

## 4.2. Loops

Loops contain a statement that is repeatedly executed as long as some condition is met. The statement that is to be executed should be placed on a separate line and indented by one tab stop.

```
while (expression)
     statement ;

for (statement(s); expression; statement(s))
     statement ;

do
     statement ;
while (expression) ;
```

## 4.3. Compound Statements

Compound statements are composed of multiple statements enclosed by curly braces. The opening brace for compound statements should be placed on a separate line, in the same column as the enclosing block. Individual statements within the compound statement should each be on a separate line, indented by one tab stop. The closing brace should also be on a separate line, in the same column as the opening brace. The following two sections contain examples.

### 4.3.1. Conditional

```
if (expression)
{
     statement ;
     statement ;
}

if (expression)
{
     statement ;
     statement ;
} else
{
     statement ;
     statement ;
}

if (expression)
{
     statement ;
     statement ;
} else if (expression)
{
     statement ;
     statement ;
} else
{
     statement ;
     statement ;
}

switch (expression)
{
case ABC:
case DEF:
     statement ;
     statement ;
     break ;
case XYZ:
     statement ;
     break ;
default:
```

```
        statement ;
        break ;
    }
```

### 4.3.2. Loops

```
    while (expression)
    {
        statement ;
        statement ;
    }

    for (statement(s); expression; statement(s))
    {
        statement ;
        statement ;
    }

    do
    {
        statement ;
        statement ;
    } while (expression) ;
```

## 5. Commenting Conventions

The importance of comments in a file cannot be over-emphasized. As previously mentioned, the use of variables and data structures, especially globals, requires explanation. Although a comment for each statement is not mandatory, it is essential that all code be thoroughly explained; this is especially true when employing machine-dependent or "tricky" code. Comments, however, may not obscure the structure of the code; unnecessarily verbose or frivolous comments that contribute to cluttered listings are discouraged.

Five classes of comments are described here. The file description comment appears once at the beginning of each file. The function description comment appears once at the beginning of each function. The other three classes are intended for general use, and may be freely inter-mixed according to personal preference; these comments are deemed to be the personal property of their author, so few restrictions are imposed.

### 5.1. File Description

One file description comment is required at the beginning of each file. This comment is intended to contain information describing the file and its contents, and has the same general form as a block comment. This comment is

divided into five sectors:

1. Identification - this sector identifies the name of the file, the module to which it belongs, its version identifier, and an indication of when it was last modified. If SCCS* is used, all this information should be automatically provided (note that the module name can be set with an admin command that contains "-fmmodule-name"). Format for SCCS files:

```
/*
 *     %M%(%F%) - %I%  (%G%  %U%)
 *
```

   Format for non-SCCS files:

```
/*
 *     module-name(file-name) - version  (date)
 *
```

2. Description - a narrative describing the file. Include a description of where and how it is used; note any special features; etc.

3. Directory - the name and a brief description of each object in the file.

4. Author(s) - the name and location of the original author(s) of the file, and of each person who has modified the file.

5. Revision History - the date, new version indicator, perpetrator, and reason for each modification of the file; not required for SCCS files.

### 5.2. Function Description

A function description comment is required at the beginning of each function. This comment is intended to contain information describing what the function does and how it interacts with other parts of the program or with the outside world. Function description comments have eight sections, as follows:

1. This section has one of the following forms:

```
/* Function:   name - english

/* Local Function:     name - english
```

* Source Code Control System - available under the UNIX operating system.

where `name' is the function name as defined in the program and `english' is a one-line english explanation of what `name' stands for. The first form should be used for functions whose scope goes beyond the file, the second for local (static) functions.

2. A narrative describing what the function is intended to do.

3. 

   * Parameters:

   followed by a list of each of the formal parameters and what they are used for.

4. 

   * Globals:

   followed by a list of global variables used and/or modified by the function.

5. 

   * Input:

   followed by an explanation of any peripheral input performed by the function.

6. 

   * Output:

   followed by an explanation of any peripheral output performed by the function.

7. 

   * Calls:

   followed by a list of function or macros that are invoked.

8. 

   * Returns:

   followed by a list of possible return values from the function.

### 5.3. Block Comments

Block comments are used when relatively lengthy explanations are required. When block comments are placed within a block of code, they should be tabbed over to the currently prevailing tab stop. A block comment begins with "/*" on a line by itself, and ends with "*/" on a line by itself. It is recommended, but not required, that each line within a block comment begin with an asterisk; it is also

recommended, but not required, that all leading asterisks be placed in the same column. Some programmers may prefer to tab or space over the text of each line in the comment.

### 5.4. Short Comments

Short comments generally fit on one line. They may be tabbed over to the currently prevailing tab stop, and may be separated from the text above and below by one or more blank lines.

### 5.5. Appended Comments

Appended comments appear to the right of the code or declarations to which they refer. When more than one appended comment appears within a block, they should all begin at the same tab stop. Appended comments may extend over more than one line; all conceivable continuation conventions are permitted.

## 6. Miscellaneous

This section contains assorted programming guidelines. Whether, and to what degree, these are applied is dependent on the requirements of the application.

1. Portability - the importance of writing portable code is a function of the application; some will require portability, others will require optimization for a particular processor or operating system. Portable code is recommended where feasible; non-portable code should be so noted in the source, and isolated from the portable code as much as is possible.

2. Embedded Statements - this is a very powerful feature of C; beware, however, of carrying their use too far. There is a trade-off between clarity and run-time efficiency that should be kept in mind.

3. goto's - goto statements are considered a necessary evil, and should be used only as a last resort. The targ ' labels for goto's should be placed on a line by themselves, beginning in column one.

## 7. Conclusion

It is recognized that the standards set forth here will not be followed in all cases. In those cases where there is a deviation from these standards, this should be noted at the point where the deviation occurs; in addition, the following text should be inserted at the beginning of the file:

```
/**************************************************************/
/***********                                       ***********/
/****              This is a deviate file                ****/
/***********                                       ***********/
/**************************************************************/
```

Following is a sample file.

```
/*      String Classification(s.strclass.c) - 1.1  (7/13/81  16:02:26)
 *
 *      This file contains functions which pertain to the classificati
 * of character strings based on the characters contained in the
 * string.
 *
 * Contents:
 *      classify  - function to determine the actual class of a string.
 *      isclass  - function to verify that a string is of a
 *                 particular class.
 *
 * Author:      Dennis F. Meyer
 *              Uniq Computer Corporation
 *              143 First Street
 *              Batavia, IL 60510
 *              (312)879-1566
 */

#include        "class.h"       /* Define classification bits */
                                /* and the global Class array */
```

```
/* Function:    classify - determine a strings classification
 *
 *      This function examines all the characters in the string in order
 * to determine the classes of characters which appear in the string.
 *
 * Parameters:
 *      s       pointer to the string to be classified
 *
 * Globals:
 *      Class   (readonly) table of classification bits for each character
 *
 * Input:
 *      none
 *
 * Output:
 *      none
 *
 * Calls:
 *      none
 *
 * Returns:
 *      integer bit table showing all classes of characters in the string
 */

int
classify(s)
register char *s ;                      /* Argument is pointer to string */
{
        register int actual_class ;             /* Accumulates bits */

        actual_class = STR_NONE ;               /* Initially no class */

        /* Repeat until end of string or all character types seen */

        while ((*s != '\0') && (actual_class != STR_ALL))
                actual_class |= Class[*s++] ;   /* OR in the next bit */
        return (actual_class) ;
}
```

```
/* Function:    isclass - see if string is of a given class
 *
 *      Check all characters in the given string to ensure that they
 * are within the given class.
 *
 * Parameters:
 *      s       pointer to the string to be tested
 *      type    bit table showing the classes of allowable characters
 *
 * Globals:
 *      Class   (readonly) classification bits for each character
 *
 * Input:
 *      none
 *
 * Output:
 *      none
 *
 * Calls:
 *      classify (strclass.c) determine actual classification of string
 *
 * Returns:
 *      The actual classification on success
 *      Error code on failure
 */

int
isclass(s,type)
register char *s ;                          /* Pointer to the string */
register int type ;                         /* Type required */
{
        register int actual_class = STR_NONE ;  /* Accumulates bits */

        if (type == STR_ALL)                    /* If all types allowed
                return classify(s) ;            /* Just get actual class

        while (*s != '\0')                      /* For each character */
                if ((Class[*s] & type) == 0)    /* if type not allowed *
                        return (CHR_ILLEGAL) ;  /* Illegal character */
                else                            /* If a legal character
                        actual_class |= Class[*s++] ;
        return (actual_class) ;                 /* Success */
}
```

# UNIX-LIKE SYSTEM STANDARDS

The UNIX* system has been proposed as a standard system for various user environments, such as personal computing, program development, and general purpose timesharing [(1), (2), (3)] . There is a substantial difference between proposing the UNIX system as a standard, and proposing standards for UNIX-like systems. To draw a correlation with languages, Cobol and Basic are highly prevalent languages for commercial applications. Cobol, however, has language standards and some substantial commonality between most implementations. Basic has minimal standards, with far too little commonality between the hundreds of implementations; yet it is a de facto standard language for small business systems.

Like Basic, UNIX systems are now running on a wide range of vendor's equipment from 8-bit to 32-bit systems. And while UNIX software is a closely controlled (though non-supported) product of Bell Labs/Western Electric, there are now multiple versions and independent implementations provided through sources outside of Western Electric. Versions include variations on "Rev 6" UNIX, PWB, "Rev 7" UNIX, and Bell's most recent System III (Microsoft, University of California-Berkeley, and others).

The development of compatible systems that are not licensed through Western Electric from vendors such as Charles River Data Systems ('UNOS'*), Whitesmith ('Idris'*) and Mark Williams ('COHERENT'*) provides additional strength to the argument for "UNIX" standards. For the purchaser seeking a vendor independent operating system in terms of both hardware and even operating system supplier, a UNIX-compatible system would seem to be one of the only choices. However, he finds no standard or basis for evaluating the correctness or completeness of a system's "UNIX-like"

nature, or relating the facilities to his application requirements.

One UNIX users group, "/usr/group" (Box 8570, Stanford, CA 94305) has formed a standards committee to address some of these issues. There is a sufficiently large number of UNIX and UNIX-like suppliers, and more importantly, enough companies looking toward UNIX-like solutions that some industry standards would be very useful.

We would suggest a standard target on these major areas:

(1) User program interface to the operating system (built around the 'C' language library--for portability). (While this is a bit language dependent, 'C' subroutine interfaces to other languages such as Pascal, Fortran and Basic are possible, and some standards here would be useful as well.)

(2) Media standards for program interchange between systems. A floppy disk standard here is most needed where interleaving and use of track Ø are issues independent of actual file structure. Nine track tape seems a little easier, but also requires interchange standards.

(3) Standards for 'tool set' functionality. UNIX-like environments provide a wide range of useful program development tools and "filters." For purposes of a standard, these tools should be categorized into functional groups.

The Cobol standard, with its multiple modules, with various levels of implementation would be a useful model to follow. This allows a user to compare implementations on a high level, and make sure the implementation he's getting has the facilities he needs. For example, some users won't need some of the development tools such as YACC or LEX, and it's clear that providing such tools is not essential to meeting some level of UNIX standard.

It is not the objective of a standard to force the burden on system developers to provide facilities that their target audiences won't need, but rather to provide those buyers with enough information to make an informed evaluation and selection. Also of high importance is a guide to application developers for developing portable code. For this reason, we would propose that a standard for UNIX-like systems follow the multiple level-multiple module standard structure of Cobol.

For example, the major modules of a UNIX-like standard might include:

I. Operating System
   A. I/O control standard streams, pipes
   B. File System (directories, protection)
   C. Process management (Exec, Fork, signals, priority, environment, etc.)

II. C Language
   (Multiple modules that correspond to other language standards would be appropriate here.)

III. Utilities and Tools
   A. Text processing (editors, formatters, unique, Grep, Diff, translit...)
   B. File manipulation (copy, sort, cat, move, compare)
   C. Program development (Make, Lint, Archive, SCCS)
   D. System management (Login, ps, ls, du)

E. Data communications (cu, uucp, ...)

F. Language development utilities (YACC, Lex, etc.)

(Other groups would be warranted.)

Levels used could be as follows:

For program development utilities

Level Ø = Null (none required)
1 = Assembler, Linker, Libraries, Debugger
2 = Diff
3 = Make, Lint
4 = SCCS

For file I/O operating system capabilities

Level Ø = Not applicable, must have Level 1
1 = Standard input and output channels, standard error channels,STDIO functions, file create & delete
2 = Extensible files
3 = Pipes
4 = Named pipes

The levels are a key to evaluation of different systems. If a function is not required by all users, then it should be in a higher level. For this reason, we've suggested that YACC and SCCS be put in a high level. However, some facilities, like STDIO, are so crucial to UNIX-like operations that they would be required (i.e.: no level zero or "null" implementation of the file system). In addition, system dependent characteristics should be identified and called out for vendor specification. This includes maximum process size, maximum file size, maximum file system size, floating point number representation and precision, etc.

In the review of this approach with others, we encountered the question - could future versions of Bell's

UNIX be non-standard? Like any other product, once a standard is created, the various versions of Bell's UNIX could be measured for compliance in terms of levels and modules. This will be just as useful for buyers as a comparison of UNIX-like systems. It is worth noting that each version of UNIX to date will have its own set of strengths and weaknesses in such a comparison. For example, IBM's PL/I is now evaluated in terms of ANSII standards.

This outline is not intended to be a detailed breakdown of modules, levels and UNIX facilities nor are the examples intended as specific proposals. However, it provides some basis for the discussion. We urge the IEEE standards group to join with the /usr/group standards group and see if a useful standard for UNIX-like systems can be established.

(1) Haynes, "UNIX--A Software Marketing Phenomenon," Computer, June 1979

(2) Isaak, "UNIX--From One Marketeer's View," Computer, November 1979.

(3) Cherlin, "The UNIX Operating System: Portability a Plus," Minimicro, April 1981

---

Jeff Goldberg
Vice President Software Development
ACM, SIGops, SIGplan, IEEE, IEEE Computer Society

Jim Isaak
Product Marketing Manager
ACM, SIGpc, SIGSmall, IEEE, IEEE Computer Society

CHARLES RIVER DATA SYSTEMS, INC.
4 Tech Circle
Natick, MA 01760

# University of York Ada Workbench Compiler

## Status Report

11 June 1982

ORGANISATION: University of York

CONTACT: Dr Ian Wand
Department of Computer Science
University of York
Heslington
York, YO1 5DD
United Kingdom

Tel: +44-904-59861 X5570
Telex: 57933 YORKUL

FUNDING: Science and Engineering Research Council of Great Britain

DEVELOPERS:
Ananda Amatya
Jim Briggs
Charles Forsyth
Chris Johnson
John Murdie
Ian Pyle
Colin Runciman
Ian Walker
Ian Wand
Tony Williams

Dave Holdsworth, of Leeds University Computing Service helped with the early development of the run-time debugger

PURPOSE: Compiler for full revised Ada, plus associated tools

HOST/TARGET: Host: VAX-11/780 running Berkeley UNIX
Target: Host or PDP-11/LSI-11 either running UNIX or "bare"

COMPLETION: End of December 1982, although this is uncertain until the implications of revised Ada are understood

IMPLEMENTATION LANGUAGE: C, although Ada has been used for the coding of some of the input/output libraries

COMPILER STRUCTURE: The compiler is in six passes which use an intermediate data structure called AIR (Ada Intermediate Representation) during all phases of the compilation. AIR comprises both an abstract syntax tree and associated

management and semantic information. The first pass is a top down syntax analyser; the next three passes carry analysis of declarations, scopes and names; the next pass performs type analysis and remaining machine independent semantic checks; the final pass is a code generator which uses a modified version of the Aho and Johnson algorithm. There is a separate compiler manager which is responsible for the maintenance of the separate compilation libraries and for controlling the sequencing of the compiler during the analysis of a file containing multiple compilation units.

COMPILATION SPEED — Approximately 700 source lines per minute (system plus user time, measured by UNIX "time") for compilation to linked executable binary on a VAX 11/780 with RP06 disks and only one level of library dependency

SIZES — The size of the compiler sources and objects together with the run-time libraries are given in Appendix A

OBJECT CODE QUALITY — The object code is compatible with and roughly similar in quality to that produced by the VAX UNIX C compiler

LIMITS — The only limits on the sizes of items during compilation are the maximum line length, which is currently set to 132 characters, and the names of compilation units which must not exceed 12 characters in length. The largest single compilation unit we have compiled to date is about 6000 lines.

LANGUAGE IMPLEMENTED — The language implemented by the present compiler is described in detail in Appendix B. We challenge other implementors to publish a detailed description of the language implemented by their compilers! After all we could simply have written "implements most of Ada apart from generics and derived types"!

OTHER USER TOOLS — Source pretty printer
Run-time debugger (early version)

AVAILABILITY — A first release will be made to UK Universities, Polytechnics and Research Council Laboratories during September 1982. This release will implement the level of language outlined below, together with generics and most remaining attributes. A further release will be made in 1983. The second release will be of a full language compiler.

Availability of the compiler to commercial users is under discussion with the British Technology Group (who

have the right to exploit any "product" developed with SERC funds).

FUTURE PLANS

An implementation on the Three Rivers/ICL Perq will start in the near future; the first release is planned for mid-1983. This implementation will run under the UNIX-like operating system for Perq under development at the SERC Rutherford laboratory. This project is also funded by the SERC.

We have two research projects which are looking at the hardware and software aspects of run-time debugging of Ada programs. The latter is using colour graphics techniques. Furthermore, we are investigating "expert system" methods for the presentation of compile-time diagnostics.

Finally we have two APSE-related projects. One is investigating command languages suitable for use in an APSE, the other is looking at the filing system structures necessary and the associated tools.

REFERENCES

J S Briggs et al: "Ada Workbench Compiler Project 1981", University of York Computer Science Report number 48 (January 1982)

C W Johnson and C Runciman: "Semantic Errors - Diagnosis and Repair", ACM compiler construction conference, Boston, USA, June 1982 (to be published in SIGPLAN Notices)

## Appendix A: Sizes of Compiler Parts and Libraries

### A.1 Lines of C in Compilation System Source Modules

#### Specifications

1759 AIR definition
2468 total of other specifications (mostly small)
----
4227

#### Compiler driver

The Compiler driver controls the compiler during single compilation unit compilations.

2431 compiler driver

#### Compiler manager

The Compiler manager controls the compiler during multiple compilation unit compilations and manages libraries of compilation units.

3884 compiler manager
217 library aquisition
76 library identity
62 convert AIR for subprogram specification to that for body
----
4239

#### Parsing and AST building

710 lexical analysis
3055 top-down parsing and building
1135 error recovery
1288 context-sensitive checks and transforms
----
6188

#### Semantic analysis

2618 dictionary building
1763 dictionary completion
5156 identification
2631 type resolution
328 name-sensitive semantic checks
-----
12496

Machine independent code generation

| | |
|---|---|
| 386 | record field allocation |
| 2385 | blocks and statements |
| 2453 | miscellany |
| 1607 | declarations |
| 1581 | expression transforms |
| 4542 | expression coder |
| 110 | target externals |
| ----- | |
| 13064 | |

Machine dependent parts of code generator

| | |
|---|---|
| 723 | templates |
| 50 | VAX machine parameters |
| 1875 | VAX code body |
| ---- | |
| 2648 | |

Abstract data type packages

| | |
|---|---|
| 113 | flattened lists |
| 196 | dynamic lists |
| 76 | sets |
| 71 | bit-packed sets |
| 52 | stacks |
| 37 | strings |
| 2429 | universal numbers |
| 277 | literals |
| 1437 | definitions |
| 857 | dictionaries |
| 468 | formulae |
| 2032 | types |
| 425 | subprogram signatures |
| 345 | static expressions |
| ---- | |
| 8310 | |

Other support packages

| | |
|---|---|
| 6061 | AIR input/output from/to linear form |
| 192 | memory allocator |
| 1855 | diagnostic message control |
| 829 | separate compilation aids |
| 246 | process AIR management nodes |
| ---- | |
| 9183 | |

```
-----
62848   TOTAL compilation system source lines
-----
```

A.2 Size in Bytes of Compilation System Object Modules

Compiler driver

```
15948   code
 4576   static initialised data
 2056   static uninitialised data
-----
22580
```

Compiler manager

```
27740   code
 8536   static initialised data
 2056   static uninitialised data
-----
38332
```

Compiler - syntax analyser

```
 90288   code
 15048   static initialised data
  2208   static uninitialised data
------
107544
```

Compiler - semantic analyser and code generator(VAX)

```
285696   code
 70656   static initialised data
 22792   static uninitialised data
------
379144
```

A.3 Lines of C/VAX Assembler in Run-time System Source Modules

Specifications

```
707   specifications
```

General run-time support library

```
477   run-time support
```

Machine independent tasking library

```
 139   clock handler
 922   tasking
  97   exception handling
  78   coroutine package
----
1236
```

Machine dependent tasking library - VAX

```
  44   array comparison (assembler)
  29   stack switching (assembler)
  49   exception handling
----
 122
```

Predefined package implementations

```
1104   package YORK_TEXT_IO
 620   package INPUT_OUTPUT
 133   package REPORT (for SofTech test cases)
----
1857
```

```
----
4477   TOTAL run-time system source lines
----
```

A.4 Size in Bytes of Run-time System Object Modules

General run-time support library

```
1552   code
 532   static initialised data
  12   static uninitialised data
----
2096
```

Tasking library - VAX

```
5900   code
1004   static initialised data
  40   static uninitialised data
----
6944
```

Predefined package implementations

The predefined packages presently comprise 'YORK_TEXT_IO', 'INPUT_OUTPUT', and 'REPORT'.

```
 9980   code
 2016   static initialised data
-----
11996



-----
21036   TOTAL run-time system object bytes
-----
```

## Appendix B: Subset Supported

The York Ada Workbench compiler supports the language defined in the Language Reference Manual of July/November 1980 [LRM], but with the qualifications described in this appendix.

Each item is preceded by the relevant section number of the LRM. Those entries introduced by "-" describe unimplemented features and those by "~" describe deviations from the LRM. An "*" is used to introduce explanation.

2.8 - Pragmas.
 * As a result of this restriction, LRM sections 11.7, 13.9 and Appendix B do not apply.

3.1 - Generic subprograms and generic packages.

3.2 - Number declarations.

3.4 - Derived types.

3.5.6 - Accuracy constraints.
 * No derived types and no accuracy constraints means no user-defined numeric types.
 * No numerics may involve <u>digits</u> or <u>delta</u>.

3.5.9 - Fixed point types.

3.6.3 ~ String literals are only of type STRING.

3.7.1 - Discriminants used as the bounds in a index constraint.

4.1 - Indexed components, selected components or slices, where the preceding name is a function call.

4.1.4 - Certain attributes. See under appendix A.

4.3 * Aggregates require a context which determines a unique aggregable type. (This is in line with the March 1982 revision).

4.3.2 - Array aggregates with named components, including others.

4.5.2 - Membership operators.

4.8 - Allocators with initial values.

4.9, 4.10 * Static expressions are more restricted than those described in the LRM; in particular, only scalar types can be involved. (This is in line with the March 1982 revision).

5.9 - Goto statements and <<Labels>>.

7.4 - Private and limited private types.

8.4 - Use clauses may only appear in context specifications. An identifier that is declared in the visible part of more than one used package is made visible: a warning is issued.
 ~ The effects of use clauses are always considered in overload

resolution. (This is in line with the March 1982 revision).

8.5 - Renaming packages, subprograms and tasks (task renaming has been removed in the March 1982 revision).

8.6 * SYSTEM is a separate package. Its definition is given in Appendix F of this document. (This is in line with the March 1982 revision).

9.6 - Type DURATION, package CALENDAR and function CLOCK are not available as part of STANDARD (but see appendix C)
* The expression following "delay" must be of type integer; it is not in seconds - it simply denotes relative time.

9.11 - Procedure SHARED_VARIABLE_UPDATE.

10.2 - Subunits of compilation units.

10.4 ~ The packages YORK_TEXT_IO and REPORT are predefined in every library and may not be redefined.

10.5 ~ A package body is elaborated immediately following its corresponding declaration.

11.6 - Exception FAILURE. (This is in line with the March 1982 revision).

12 - Generics.

13.1 - Length specifications.

13.6 - Change of representation.

13.7 * Package SYSTEM is defined under Appendix F.

13.8 - Machine code inserts.

13.10 - Unchecked programming, via conversion and deallocation.

14 - Packages INPUT_OUTPUT and TEXT_IO.
* We provide only a limited form of text input/output. This is done using the package YORK_TEXT_IO which provides a subset of the facilities in TEXT_IO.

Appendix A - Attributes IMAGE, VALUE, POS, VAL, PRED, SUCC, DELTA, ACTUAL_DELTA, BITS, LARGE, MACHINE_ROUNDS, DIGITS, MANTISSA, EMAX, SMALL, LARGE, EPSILON, MACHINE_RADIX, MACHINE_MANTISSA, MACHINE_EMAX, MACHINE_EMIN, MACHINE_ROUNDS, MACHINE_OVERFLOWS, RANGE and STORAGE_SIZE are unimplemented. PRIORITY and FAILURE are unimplemented in line with the March 1982 revision.

Appendix C - The version of package STANDARD supplied omits the following:

SHORT_INTEGER
LONG_INTEGER
SHORT_FLOAT
LONG_FLOAT
package ASCII

```
subtype PRIORITY
type DURATION
package CALENDAR
generic procedure SHARED_VARIABLE_UPDATE
generic procedure UNCHECKED_DEALLOCATION
generic function UNCHECKED_CONVERSION
generic package INPUT_OUTPUT
package TEXT_IO
package LOW_LEVEL_IO
```

Packages ASCII, YORK_TEXT_IO and CALENDAR are available as separate library units.

Appendix F (VAX and PDP-11)

1) All pragmas are accepted, but ignored.

2) No implementation dependent attributes are available.

3)

```
package SYSTEM is

    type SYSTEM_NAME is (PDP11, VAX, INTERDATA, M68000, PERQ);
    type OPERATING_SYSTEM is (UNIX, GCOS, IBM, RSX);

    NAME: constant SYSTEM_NAME := VAX;
    OS_NAME: constant OPERATING_SYSTEM := UNIX;

    STORAGE_UNIT: constant INTEGER := 8;
    subtype ADDRESS is INTEGER;

    TICK: constant INTEGER := 1;

    MIN_INT: constant INTEGER := -2_147_483_648;  -- -2**31
    MAX_INT: constant INTEGER := 2_147_483_647;   -- 2**31-1
    MAX_DIGITS: constant INTEGER := 7;

end;
```

4) The system storage unit (used in representation specifications) is the 8-bit byte. Fields that are records or floating point numbers may start and end on byte boundaries. Scalar fields may start and end on arbitrary bit boundaries. If representations specifications are used to specify the components of a record, all components must have representations supplied.

5) There are no system dependent component names.

6) Address specifications are interpreted as byte addresses of the named entities.

7) UNCHECKED_CONVERSION is not implemented.

8) A main program must be a parameterless procedure.

UNIVERSITY OF YORK

Department of Computer Science

8th September 1982

## An Ada Compiler for VAX UNIX

The Software Technology Research Centre of the Department of Computer Science at the University of York announces the availability of a preliminary release of its Ada Workbench compiler from 15 October 1982. The compiler is available, in object form only, free of charge to UK educational establishments and Research Council Laboratories.

The compiler:

* supports the Ada language of July 1980, but excludes generics, derived types, private types, subunits, number declarations, and accuracy constraints. A complete Ada compiler will be released in 1983;

* produces good diagnostic messages which can be made verbose or very verbose for the beginner;

* can be hosted by Berkeley 3 and 4 UNIX and UNIX 32v on the VAX-11;

* compiles Ada source code at approximately 700 lines per minute on a VAX-11/780;

* produces good quality code for the VAX-11 which runs in conjunction with a small run-time system.

The distribution package consists of:

- Compiler executable code
- RTS executable code
- Compiler manager executable code
- Installation Guide
- Functional Specification
- Test cases

The completion of a Software Licence and a BTG Confidentiality Undertaking is required before the compiler can be supplied.

Please note that the compiler cannot (and never will) be hosted on a PDP-11.

---

Ada is a registered trademark of the U.S Government, Ada Joint Program Office.
VAX is a trademark of the Digital Equipment Corporation.
UNIX is a trademark of Bell Laboratories.

## Ada Workbench Compiler - Distribution Questionnaire

If you are interested in obtaining the compiler, please complete this questionnaire and return it to:

"Ada Compiler Distribution",
Software Technology Centre,
Department of Computer Science,
University of York,
Heslington,
YORK,
YO1 5DD,
UK.

---

Name of Department, Establishment:

____________________________________________

Address:

____________________________________________

____________________________________________

____________________________________________

____________________________________________

Contact:

____________________________________________

Telephone:

____________________________________________

Please tick the following options where applicable:

Hardware: VAX-11

730 _____ 750 _____ 780 _____

Store capacity:

__________ Mbyte

Operating System: UNIX

Berkeley 3 _____ Berkeley 4 _____ 32v _____

Preferred distribution medium: 1/2" 9 track magnetic tape

800bpi _____ 1600bpi _____

# The Loosing of the Sicky bit, or How to Speed up Your UNIX.

"What on earth is the sticky bit"? A plea uttered by more than one user of a Unix system, I know from experience. If you're one of them, then read on and find out what it is. Even if you aren't, read on, because later in this piece I intend to show that the damned thing is a positive nuisance. If you feel adventurous, and have a source code licence, you might even do what I've done: abolish it, and get a 30-40% improvement in the response of your system.

## What it is, and excuses for why.

As is commonplace nowadays, Unix overspends its budget - give it n Kbytes of main memory and it will happily allow you to create enough concurrently running programs (processes) to use far more than that. To make up for the lack of real memory, the Unix kernel uses space on one of the filestore discs to extend the apparent memroy available. For a process to run it has to be located in real memory, but if it is suspended for any reason (waiting for i/o, used up its share of mill time....) then it can be written out onto disc. Writing it out releases real memory, so a process already written out but is now ready to run can be read in and allowed to continue. The kernel has the job of making all this invisible to the programmer, which it does. The whole business of transferring programs between disc (secondary memory) and real (primary) memory is known as swapping and is a well known feature of anything that deserves to be called an operating system [1]. CP/M users please note.

To reduce the amount of real memory required by a typical workload - say everybody having the shell waiting, about half the users deep in 'ed' and the rest doing more or less random things - the kernel allows programs to share memory. If five people are using the editor then it doesn't make a lot of sense to have five totally separate copies of the editor, one each, shuffling to and fro between disc and memory. Provided that the editor doesn't indulge in antisocial practices like modifying itself as it runs, there's no reason why the five users shouldn't share one copy of the program between them. They will need private data areas, obviously, but the instructions part of the editor will be the same for them all.

By sharing the instructions part of each program between a number of processes, the amount of swapping can be reduced. The system will only swap stuff when it has to; to make memory available for a process already swapped out that now wants to run. Reducing the demands on memory will therefore reduce swapping, which is a rather slow operation, and in this way will improve the response seen by users. Taking advantage of the fact that the shared part never changes also helps, as we shall see.

To allow instructions to be shared between processes, no process may be allowed to write into the shared part. To enforce the rule, rather than simply express the hope that it will be obeyed, the kernel puts these shared sections (known as 'texts') into write-protected memory. That won't be a hard job if the memory management hardware is adequate. For every text currently in use the kernel keeps a single copy in the swap area. If any processes needing that text are in real memory, then a copy must also be in memory - it will remain in real memory as long as any process is using it. If every process using the text happens to be swapped out, then the real memory copy of the text can simply be abandoned: it can't be different from the copy in the swap area because it has been write protected. If any process needing the text is to be swapped into real memory, then the text must also be swapped in. Not having to swap the text portion out to disc reduces the amount of swap i/o that has to be done.

All of the standard Unix compilers produce programs that can be run in this way, although the loader 'ld' has to be warned that the feature is actually required. It shuffles the program around so that the invariant parts are all in one lump and makes a note within the executable binary file that this has been done. For example;

```
    cc fred.c
```

will produce an 'a.out' that isn't shareable. Giving the '-n' flag, which 'cc' passes on to the loader, is what you want to do:

```
    cc -n fred.c
```

There is no point at all in doing this for small programs that are rarely used. It's only worth the effort if programs are large and there is a high probability that several copies will be run at the same time.

As a word of warning, this only applies to compilers that produce directly executable binary programs. If they produce an intermediate code, like the Berkely 'pi' Pascal compiler, or CIS COBOL, then it's the interpreter and not the intermediate code that should be shareable.

**********************************

The Sticky Bit.

The sticky bit - much misunderstood - now comes into play. If an executable binary file has the 'sticky' (sometimes called the 'save text') bit set in it's permission field, things change a little. In the normal run of things, that swapped-out copy of a text is junked as soon as the last process using it decides to terminate. If it's needed by another process later, it has to be re-read from the executable file and re-established on the swap disc. Unix is not well-known for the speed of its file i/o, so for a large program this can be relatively time-consuming; with big programs it can account for a substantial part of the delay a user notices between giving a command and seeing it start to run. Setting the sticky bit prevents the swap copy being

junked.

With the sticky bit set, once the swap copy of a text is set up, it remains there until the machine is re-started. The first time the program is executed the same delay will be present, but after that the text is always available to be swapped in when it's needed. Swapping is usually accomplished in a single i/o operation, whilst reading a large executable file may take several hundred. The resulting speed up may be an order of magnitude and is certainly noticeable, particularly since other processes doing their own disc i/o don't get clogged up by the unnecessary extra disc transfers.

So, if it's so good, why not set the bit on every executable file you've got? Well, there's a price to pay. All those swap disc copies of text segments absorb space, and one of the quickest ways to bring a Unix system to its knees is to exhaust its swap space. Even if you can afford enough disc to cope with it all, the kernel's internal table will overflow instead and you end up going up the ladder only to come back down the snake.

The rules adopted by most users come into one of two categories. The "I don't understand it" bunch take the easy way out and ignore the sticky bit altogether; this is not a bad thing to do. Those who do have some idea about it usually set it on the binaries they think will be heavily used. This is not a very good idea.

There's no point in setting the sticky bit on things like the shell and the editor. Loading the first copy of each will be slow anyway, and from then on the likelihood is that at least one copy will always be active, resulting in a swap copy of their text anyhow. Now, (say) the C compiler probably <u>would</u> benefit from the sticky bit. It's big, fairly regularly run, but unlikely to maintain a permanent presence except in very big systems. Short C compilations can be dominated by the time taken to load the various compiler passes, the assembler and the loader. The trouble is that it's a pain to find out the names of all these things and set the bit, so few people bother themselves about it. Incidentally, if you thought that '/bin/cc' was the name of the C compiler, you're wrong - its just the management program that invokes all the other things.

To sum up: The sticky bit is more trouble than it's worth. Forget it.

*********************************************

## One Step Beyond - Greasing the Sticky Bit.

What's wrong with the sticky bit is that it's up to the user to set it. Surely the system itself is in a better position to know

what's going on, and ought to adjust accordingly? Correct!

If you could persuade the kernel to hang on to the last so-many texts used, the most heavily used would automatically be in there when the were wanted. Experiments on a PDP11 system at Bradford University bear this out.

The experiment meant modifying the existing kernel to implement a cache of most recently used text segments, even when they became disused. A list of out of use texts is kept in most recently used order. Whenever a text is needed, a search is made of this list to see if it already exists in swap space. If it does, it is removed from the 'free' list, placed on a 'busy' list and used immediately. If it doesn't already exist, then it is dragged in from file as before. If a text is only used once, it will gradually be pushed down the free list as other texts are brought in and used, and unless the free list is very long it will eventually fall off the end. When it does, the swap space is freed.

Swap space is also freed if it runs out. When that happens, texts are thrown away in reverse order starting from the far end of the free list. This solution is much better than the standard V7 Unix approach, which is for the kernel to throw up its hands in horror.

The modifications described above have added about 1/2 Kbytes to the kernel, dispensed with the sticky bit altogether, and result in the typical time taken to compile and optimise a 50 line C program falling from 25 to 17 seconds. Users have (without prompting!) commented on a 'noticeable improvement' in response. Enough said?

Mike Banahan.
15th. December 1982.

Reference:
Fundamentals of Operating Systems,
A. M. Lister,
Macmillan,
1981,
Chapter 5.

# An ONYX implementation of an allocation checking technique.

Tony Cornah

MRC/SSRC SAPU,
Dept of Psychology,
The University,
Western Bank,
Sheffield,
S10 2TN.

ABSTRACT

This note describes the implementation of the tool originated by Barach, Taenzer and Wells[1]. That paper set out a system for detecting storage allocation errors in C programs: basically an error report for each unfreed allocation or the release of an unallocated area. However the details of how to trace the allocation and release of dynamic storage presuppose access to the source of the C library: a false supposition for holders of binary licenses. Renaming the original routines in the library, writing new ones which call the renamed ones and adding these to the library will overcome the problem.

## 1. Renaming the original routines.

A program, "rename", to copy "/lib/libc.a" to "newlibc.a" was written fairly easily: the names of the archive members which needed altering were discovered by scanning a table of contents for instances of "*alloc*" and "*free*". In fact only "malloc.o" turns up, something which may have been expected from the layout of the section of the manual dealing with the library. A namelist of the extracted "malloc.o" revealed "_malloc", "_free" and "_realloc". The last calls the first two so needs no modification (ascertained by trial and no error). The program "rename" scans for the required archive member and then scans within for the required symbols. The names are changed to "zzmalloc.o", "_zzalloc" (because of the restriction to seven characters) and "_zzfree".

## 2. The new routines.

Routines along the lines of the ones in the original paper were written. Modifications were

a) "writerecord" writes out one less return address: that of "malloc" or "free".

b) "malloc" calls "zzalloc" and "free" calls "zzfree" to perform the heap manipulation.

c) "malloc" looks at the word before the allocated area to determine the size "zzalloc" has decided to give (this word points to the last word of the allocated area in the ONYX implementation): another example of trial and error to overcome the lack of source code.

"ar" was used to add the new routines into "newlibc.a" which was then moved to "/lib/libtrace.a", where it can be used by specifying "-ltrace" in a call to "cc".

3. The analysis program, "plumber".

Again, this is slightly different from the program "prleak" described in the original paper.

a) The error reports are not sorted, nor are they counted and no average size is produced. The format of the reports differs in minor details.

b) An option, "-s", will produce a file, "summary.out", which contains the total size of unfreed storage and each unique traceback path (sorted this time!!). The total size is useful in that a working program may be stopped after all allocations have been done (or anywhere else for that matter) and "plumber" can report how much of the heap has been used. The unique traceback path reports are useful if the program being examined is likely to use each call to "malloc" or "free" very often. Reams of paper detailing all instances of a few calls are less useful than the calling sequences themselves.

c) The call to the macro "checkleaks", which calls "plumber", can specify a series of octal addresses. If this is the case then instead of reporting errors in the normal way, the program reports all instances of the allocation or release of these addresses. This is useful if the errors are duplicate frees. Having found that such an error exists then how the particular block was allocated and freed is useful information.

4. Distribution.

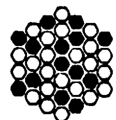
Anyone requiring a copy of this software should send me an ONYX tape cassette onto which I will "tar" the files: the C sources involved, the library "libtrace.a" and a manual page. If this is not possible then I will send listings (not of the library) or will try to prepare another transfer medium: we have a TERAK (8" single-sided, single-density, soft-sectored floppies), an Apple (Apple DOS 3.3 and 5 1/4"" single-sided, single-density floppies) and a North Star (CP/M and 5 1/4" single-sided, double-density floppies "Lifeboat P2 format"). All of these can communicate to a greater or lesser extent with our ONYCES.

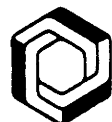
References.

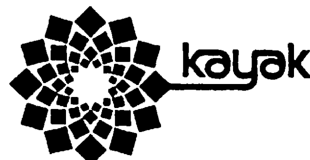
[1] Barach, D.R., Taenzer, D.H. and Wells, R.E., "A Technique for Finding Storage Allocation Errors in C-language Programs", SIGPLAN NOTICES, Volume 17, Number 7, July 1982.

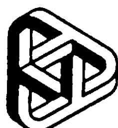
BIB 507.9
October 1982

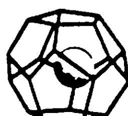
# ABSTRACTS



THE ATTACHED LISTING PRESENTS ABSTRACT OF SELECTED REPORTS
PRODUCED WITHIN THE CYCLADES, SIRIUS, KAYAK, SOL,
NADIR AND RHIN PROJECTS, AND SOME OTHER USEFUL REPORTS

ISSN 0152 - 3635

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
Domaine de Voluceau - Rocquencourt

GAL 1.502 - GE, French

GIEN M. - Pour une approche globale des problèmes de portabilité, Journées d'Etudes " Génie Logiciel", Perros-Guirec, (Janv.80), 18 p.

Software portability is far from being natural. Sources of incompatibilities that make programs non portable are numerous. Progress have been made through the use of high level, machine independent, programming languages, but operating system interfaces and lack of standardization in language implementations are still stumbling blocks of program portability. A global approach has been taken in the pilot project SOL to bring solutions to this problem. It involves standard implementations of a standard programming language (PASCAL), the development of a portable operating system (also written in PASCAL), and of a portable programming environment.

LAN 1.504.1 - TE, English

GIEN M. - A PASCAL Validation Office, European Workshop on Industrial Computing Systems (Purdue Europe) , 1980 Spring Meeting, Vienna, Austria, (Apr.1980), also in Computers in Industry, n°2, (1981), pp 109-113 .

A PASCAL validation office is presented as a key element in the development of portable software. A number of software portability issues are briefly surveyed as well as problems involved by the definition of programming language standards. The validation office is used for both the technical aspects of compiler testing and the organizational aspects of validation procedures.

LAN 1.513 - TE, French

MAURICE P.et AL. - Spécifications du langage intermédiaire (LI) généré par la souche commune de traducteurs PASCAL (Version 1), (Déc.80), 39 p .

This document describes the Intermediate Language (LI) used to express the code produced by the common trunck of the SOL Pascal Compilers.

This Intermediate Language is to be translated into machine code by a code generator for each given target machine.

LAN 1.538.2 - TE, French

MAURICE P. - Complément de spécification à la norme du langage PASCAL et mise en oeuvre dans les processeurs PASCAL-SOL, (Nov.81), 18p.

This note defines complements to the standard specifications of the programming language PASCAL (ISO 7185) in order to reduce implementation choices and define standard compiler operation. Associated with the ISO document, it forms the standard external specification of the set of PASCAL compilers being developed in the SOL project. Validation and certification procedures will be based on the complete specifications.

LAN 1.542.1 - TE, French

MAURICE P. - Manuel d'Utilisation des Traducteurs PASCAL-SOL, (Mai 82), 74 p.

This manual is a PASCAL Users'guide , corresponding to the standard SOL specifications for PASCAL compilers based on the ISO standard. Its structure is inspired by the "PASCAL Report" and "PASCAL User Manual" by Wirth and Fensen-Wirth. It aims at familiarizing the reader with peculiarities of the SOL compilers.

Elements which remain implementation dependent are indicated; they may be detailed in complementary manuals provided with any specific compiler.

LAN 1.554 - TE, French

BRZEK N./ ESTEVE P./ FORTIER R./ JAOUA A./ MAURICE P. - Souche Commune des Traducteurs PASCAL-SOL, (Mars 81), 9 p.

This note is a general presentation of the SOL common trunk for PASCAL Compilers. It is made of two passes, the last one generating an Intermediate Code to be processed by a machine-dependent code generator. It is entirely written in PASCAL and processes the language as defined by the standard SOL specifications, based upon the complete ISO standard. It is freely available through the Club SOL.

LAN 1.559 - TE, French

D'AUGSBOURG B./ MAURICE P. - Optimisation de Traducteurs: Application à la Souche PASCAL-SOL. Univ.Paul Sabatier, Toulouse, (Mai 82), 9 p.

This note is an overall presentation of the optimisation method followed in order to produce optimised code from the SOL common trunk of PASCAL Compilers. Optimisations are applied on the Intermediate Codes produced by the common trunk and are independent of the trunk itself.

MAT 1.509 - TE, French

FINGER U./ MEDIGUE G. - SM90: Une architecture Matérielle Modulaire et Ouverte, (Apri.82), 15 p.

SM 90 is a multi-microprocessor architecture designed to take the advantage of the various 16-bit microprocessors available (68000 and 16000 in particular). It is built around a common bus designed with reliability in mind. Each processor board accesses also a local bus. Memory boards can be attached to either bus(independentely or simultaneously) according to sharability or speed requirements. I/O are performed through slave processors attached to the common bus.

This architecture is used in particular to implement the SOL operating system as well as the CHORUS distributed system kernel.

ORG 1.529.1 - GE, French

GIEN M. - Le Club SOL, (Fév.82), 3 p.

The purpose of the"Club SOL" is to serve as a forum for organizations using or providing products conforming to the SOL Standards (regarding PASCAL or the SOL operating systems interface).

It is the means to obtain the machine independent trunks developped by the SOL project for PASCAL Compilers and the SOL portable operating system .

It should maintain and insure the evolution of the SOL standards and the associated trunks after the project is terminated.

ORG 1.547 - GE, English

GIEN M. - General Presentation of the SOL Pilot Project, (June 82), 8p.

The SOL pilot project is developping a basic software engineering environment portable and written in PASCAL, intended for supporting research , development and operation of software engineering tools. It is based on standard PASCAL compilers, a Unix compatible operating system, written in PASCAL, and a set of software tools, also written in PASCAL. It is implemented on a variety of mini and micro-computers. Validation procedures are applied to any particular implementation to check their conformity with the SOL standard interfaces, thus insuring real portability to any program based upon these interfaces.

RES 1.501 - TU. French

GIEN M. - Introduction aux Réseaux Locaux, (Apr.79), 11 p .

This paper is a short tutorial introduction to Local Area Networks. It gives an overview of the characteristics of local networks and of their various components :

- Local Networks Architecture
- Transmission media
- Configurations
- Communication procedures
- Protocols.

SYS 1.504 - TU, French

GIEN M./ GUILLOT J.M. - Présentation des concepts du système UNIX*, Bigre N°16, (Nov.79), 18 p.

The pilot project SOL aims at developing on French mini or micro-computers a program production environment equivalent and even superior to those which can be found on the best American minis.

This paper describes some of the main features of the UNIX system which could be taken as a model for the systems support of a good software production environment.

*: UNIX is a trade mark of the Bell Laboratories.

SYS 1.511 - TE, French

HANOUT J.C. - SYSPAS : un mini-système écrit en Pascal, (CNAM), (June 82) 27 p.

SYSPAS is an experimental multi-user system written in PASCAL. The compiler used is the OMSI-PASCAL compiler under RT11 running on a DEC-LSI-11/03. Bootstrapping is realized through RT11.

A few concepts were added to PASCAL in order to structure the system : process, module and semaphore. The module concept implements the concept of abstract data type. The process concept allows the handling of several terminals and peripheral devices.

Some external procedures were written in MACROLL to implement the structuring concepts and interface the hardware.

SYS 1.530.2 - TE, French

BON C./RACHELSON P./THIRE B. - Assembleur/Editeur de Liens SOL : Spécifications externes, (Sept. 81), 31 p.

This document describes the external specifications of the machine adaptable Assembler and Link Editor associated with the SOL operating system. The assembler processes the native machine instruction set but provides standard machine independent directives. It is to be very simple and may be used as the last pass of a compiler.

SYS 1.543 - TE, French

CAMPBELL I./CHEMLA P./GIEN M./OLLIVIER G. - Le ystème d'Exploitation SOL, ( Jan. 82), 24 p.

The SOL operating system is to be used as a standard basic software engineering environment for supporting software tools research and development. It is made of a portable time-sharing kernel, written in standard PASCAL and a set of basic utility programs also written in standard PASCAL. The kernel interfaces as well as the utilities are completly compatible with the UNIX* operating system interfaces and utilities. It is implemented on a wide variety of 16 bit mini and micro computers such as Honeywell Level 6, Sems/Mitra, 68000, 8086 and 16000 based micro-computers.

Its main characteristics are its portability on various hardware, simple and standard ways for exchanching information between programs files and devices, a powerful command language allowing easy manipulation of user programs.

*UNIX is a trademark of Bell Laboratories.



UTI 1.506 - TE, French

SINZ P. - Le Langage deccommande du Système SOL et son Interpréteur. Journées Bigre 82, Grenoble, Bigre n° 28-29, (Jan. 82), 99-112.

The SOL system command language has been widely inspired by command languages available with the UNIX system. It is a superset of version 7 Shell and is compatible with it. It includes also some of the facilities provided by other shells such as the Berkeley shell. The SOL shell is a user program, written in Pascal. Therefore, it is not necessarily the only one available with the system. It has been designed and implemented so that it can be easily modified to process variants of the command language that can be better adaptated to specific usage.

TOGO INSTITUTE OF TECHNOLOGY

P.O.Box 5375 Tokyo International
Tokyo 10031 JAPAN
03(813)8991

August 17 ,1982

European UNIX Users Group
c/o Jim Mckie. Unix Support Officer
Edinburgh Regional Computing Centre
20 Chambers Street Edinburgh Scotland

Dear Manager:

Please send me more information on the unix users group.

Sincerely,

地頭重人

Shigehito Jitoh
Research Assistant
Computer Science Department

ISTITUTO DI MATEMATICA APPLICATA

FACOLTÀ DI INGEGNERIA • UNIVERSITÀ DI BOLOGNA

VIA VALLESCURA 2 • 40136 BOLOGNA • ITALIA
TEL. (051) 331588

August 19, 1982.

UNIX Newsletter.
c/o J. McKic
Edimburgh Regional Computr Center
c/o EdCAAD
20 Chambrers Street
Edimmburgh, Scotland.

Sirs:

Please enter my subscription to EUUG Newsletter starting from now and bill me.

Enclosed you can find copy of our licence for UNIXV7 and preliminary version of an article decribing what we are doing here in Bologna.

Thank you very much.

Sincerely

Luigi Cerofolini
Associate professor.

Kazuhiko Nishioka
Technical Manager
37F Sumitomo Bldg.
2-6-1 Nishi Shinjuku
Shinjuku, Tokyo 160-91
JAPAN

Attn: Jim McKie
UNiX Support Officer
European UNiX User Group
Edinburgh Regional Computing Centre
c/o EdCAAD 20 Chambers Street, Edinburgh
Scotland

Dear Mr. McKie,

We came to know your activities in "A User's Guide to UNiX Systems".

Though we are in commercial world, I want to get your newsletter if possible.

So please send your newsletter or let us know how we can read it.

Our primary interest is on the software tools on UNiX mainly for microprocessor software development.

Enclosed is the check for airmail charge.

Sincerely,

Kazuhiko Nishioka
Technical Manager

Oct 17, 1982

a/s KONGSBERG våpenfabrikk
Postbox 25 N-3601 Kongsberg. Norway
Tel. National (03) 73 82 50
International (+47 3) 73 82 50
Telex 11491 vaapn n
Telegram Våpenfabrikken
Bank Andresens Bank A/S
Den norske Creditbank

Jim McKie
c/o EdCAAD
20 Chambers Street
Edinburgh, EH1 1JZ
United Kingdom

DERES/YOUR/IHRE REF VÅR/OUR/UNSERE REF DATO/DATE/DATUM

01 Nov 1982

Dear Jim,

Thought I'd let you know how we got on with the installation of the EUUG small machine tape on our 11/34. The answer, I'm afraid, is "not very well"! The reason we were particularly interested in the system is that we wanted to replace our existing DZ/11 with an Able DHDM, and discovered that this pushed us over the address limit, by some considerable number of bytes. (We have 2 RM disks, a TU16 magtape, plus the DHDM).

The first attempt to put the EUUG system up worked, and things looked promising until we unleashed the users, when it rapidly became apparent that we weren't supporting enough processes. A comparison of the old and new param.h files quickly showed that this was correct, so we adjusted appropriately, recompiled, and ... miles too big!! To cut a long story short, we learned a lot about our original system that weekend - it turns out that it has been through the "VU" tuning mill before it reached us, and has inodes ported out, enabling considerably increased values to be used in param.h.

In the end all went reasonably well - we incorporated some extra "byte slicing" techniques from the EUUG source into our original, and managed to squeeze it back down to a sensible size again, with all the required devices, and the enough processes to sensibly support our 7-10 users. The command sources installed are now EUUG (slightly modified to handle tty00... and not ttya..., and to allow user names with non alphameric characters again - the Norwegian alphabet has 3 additional vowels over the English one, placed after z in the Ascii character set).

To give you an idea, this is what you can achieve!

```
/* KV tunable variables */
#define NBUF     25      /* size of buffer cache */
#define NINODE   125     /* number of in core inodes */
#define NFILE    100     /* number of in core file structures */
#define NMOUNT   4       /* number of mountable file systems */
#define MAXUPRC  25      /* max processes per user */
#define CMAPSIZ  30      /* size of core allocation area */
#define SMAPSIZ  30      /* size of swap allocation area */
#define NPROC    60      /* max number of processes */
#define NTEXT    30      /* max number of pure texts */
#define NCLIST   65      /* max total clist size */
```

åsKONGSBERG våpenfabri

```
/* original tunable variables */
#define NBUF     8          /* size of buffer cache */
#define NINODE   75         /* number of in core inodes */
#define NFILE    75         /* number of in core file structures */
#define NMOUNT   5          /* number of mountable file systems */
#define MAXUPRC  15         /* max processes per user */
#define MAPSIZ   (NPROC/2)
#define NPROC    50         /* max number of processes */
#define NTEXT    25         /* max number of pure texts */
#define NCLIST   100        /* max total clist size */
```

However ... I have the distinct impression that the system is now slower than it was before, which is rather puzzling. Unfortunately, we almost simultaneously had a disk head crash, with the result that we have reorganised our file system placement, and are not yet certain whether or not this is the cause of the performance change. I heard you had a system in Edinburgh with a DHDM in - any ideas?

On the question of performance, I have one tip to pass on to people using uucp/uux a lot. Uucp searches the spool directory to find if there is work to be done, building up a table of the files it finds, emptying this table, and then searching again. IF the spool directory has been very full at some point, then this search can take a long time (because a directory never shrinks, but simply gets null entries). Under heavy load, we have found out that the other machine simply times out waiting for a request acknowledgement (usually HY). The obvious solution is to increase the MAXMSGTIME parameter, but if you cannot persuade your correspondent to do this, then a "quick and dirty" fix is simply to delete and recreate the spool directory.

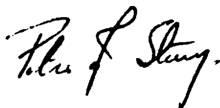
Which brings me to the next point. Hugh Connor wonders (EUUG Newsletter, Vol 2, No 1, p71) why anyone would want to join a line to itself with ed. Unless anyone can show me a neater way to do this, the answer is in this operation:

```
g/^$/.,/./-1j
```

which turns all sequences of one or more empty lines in a file into a single empty line. So my preference is not to put in his change!

I don't know what kind of editorial policy you follow as regards what is "useful" and what isn't, so you can decide yourself whether or not to include the attached shell procedures, which are designed as an aid to system managers tired of sending pleading mail about disk space. We have used them with considerable success here, and are pleased with the response from the users, who tend to "cooperate" with the setup.

Yours sincerely
A/S Kongsberg Vaapenfabrikk
Avdeling U4



Peter J Story

This procedure will search from the specified directory for "junky" or "big" files, and make a set of mail files to be sent to the owners. There is an "ok" lists of junk and big files which are not notified, so that the users can send the names of "wanted" files back in the mail, and avoid being pestered in the future.

```
: Synopsis:   spacefinder directory

: finds all files from the given directory which look junky or big
: and makes 'mailowner' files
: These can be examined, and then mailed to the owners with
: the procedure 'sendmail'

: Bug if too many files - ls just gives up!

if test $1
then
   FOUND=/usr/tmp/jnka$$
   NOTOK=/usr/tmp/jnkb$$
   LS=/usr/tmp/jnkc$$
   trap 'rm $LS $FOUND $NOTOK; exit;' 1 2 3 15
   find $1 \( -name '*junk*' \
      -o -name '*tmp*' \
      -o -name '*temp*' \
      -o -name '[a-lnoq-zA-Z]' \
      -o -name 'core' \
      -o -name 'nohup.out' \
      -o -size +50 \
      \) -print \
      | sort >$FOUND
   if test -s okjnkbig
   then
      comm -13 okjnkbig $FOUND >$NOTOK
      rm $FOUND
   else
      mv $FOUND $NOTOK
   fi
   if test -s $NOTOK
   then
      ls -ld `cat $NOTOK` \
      | tee $LS \
      | sed -e '/not found/d' \
           -e 's/^.............//' \
           -e 's/ .*//' \
      | sort -u \
      | while read owner
        do
           led $LS >/dev/null <<?
              v/^.............$owner /d
              W mail$owner
              q
?
        done
      rm $LS $NOTOK
   else
      rm $NOTOK
   fi
else
   echo 'Synopsis: jnkfinder directory'
fi
```

I prefer to scan the files before I send them, so I have the second procedure available for doing that.

```
: takes all files starting 'mail', and mails them, preceded by the
: current 'hdr' file, to the person named in the rest of the
: mail file name
: eg, mailpete results in 'cat hdr mailpete | mail pete'

: designed for the output files from procedure jnkfinder, bigfinder
: etc

ls mail* \
    | sed -e 's-^....\(.*\)-cat hdr & | mail \1 \&\& rm mail\1-' \
    | sh
```

**TITN** TRAITEMENT DE L'INFORMATION TECHNIQUES NOUVELLES

SIÈGE SOCIAL ET BUREAUX :
1 à 5, RUE GUSTAVE-EIFFEL, 91420 MORANGIS
TÉL. : 909-34.44 TÉLEX : 691163 TITN MOR

*Chilly, le 2 Novembre 1982*

N./Réf. : *DJ/cd - 82.261/A*
V./Réf. :

*UNIX NEWSLETTER*
*C/O Jim McKie, Unix Support Officer*
*Edinburgh Regional Comuting Centre*

*C/O EDCAAD*
*20 chambers St*

*Edinburgh*

*SCOLTAND*

*Dear Sirs,*

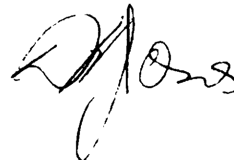
*TITN, a large OEM which is part of the THOMSON Group, is interested in UNIX related products and information.*

*Would you please send us information about your newsletter so we may consider subscribing.*

*Regards.*

*F. TARTANSON*
*Ingénieur en Chef*

*D.H. JONES*
*Ingénieur*



AGENCE : TITN - PROVENCE
7, RUE LOUIS-ARMAND
Z.I. D'AIX EN PROVENCE
13763 LES MILLES CEDEX
TÉL. : (16.42) 26.37.49 - TÉLEX 400 221

S. A. au capital de 990.000 F R. C. CORBEIL 662 024 983 B SIRET 662 024 983 000 30 Code APE 7703 C. C. P. PARIS 2.575-09

# EUUG

European UNIX® Systems User Group

Hugh Conner
Dept. of Electrical and
Electronic Engineering
Heriot-Watt University
Edinburgh EH1 2HT

Dear Jim,

Some information on a known bug which I thought might be worth printing. Anyone who has the EUUG distribution system will know of a problem on 11/44s and 11/23s with long division. A fix for this has already been published and requires ifdefing some code with DIVFIX. Well, it now appears that these are not the only machines which require this fix. I was recently informed that on 11/34s the division of 32768 by 1 gave the answer 32767. Tests on our 11/34 showed that it requires DIVFIX as well. I advise others to try this. Also, if anyone has an 11/60 could they see if DIVFIX is required for that machine. It may be that the older machines (11/40, 11/45, 11/70) are okay, while the newer models (11/23, 11/24, 11/34, 11/44, 11/60) all require the fix. Actually I think that one solution may be to always include the DIVFIX code, as it still works on machines that don't require it.

Yours sincerely,

Hugh M. Conner

P.S.

For DIVFIX read DIV_FIX

**EXECUTIVE COMMITTEE**
*Chairman*: EMRYS JONES, CGram Software
*Meeting Secretary*: PETE COLLINSON, University of Kent
*Membership Secretary*: HUGH CONNER, Heriot-Watt University
*Executive Editor*: JIM McKIE, Edinburgh Regional Comp. Centre
*Newsletter Editor*: CORNELIA BOLDYREFF, SW Univ. Regional Comp. Centre
*Newsletter Editor*: BRUCE ANDERSON, University of Essex
*Member*: NIGEL MARTIN, University College London
*Member*: MIKE BANAHAN, Bradford University

®UNIX is a trademark of Bell Laboratories

SC METRIC A/S

SKODSBORGVEJ 305 . DK-2850 NÆRUM . DENMARK . TEL. (02) 80 42 00 . CABLE METRICTRADE . TELEX 37163 . A/S REG. NR. 37994

EUROPEAN UNIX USER GROUP
Edinburg Regional Compating Centre
c/o Chambers Street
Edinburg, Scotland.

YOUR REF.: OUR REF.: KBN/ksh NÆRUM December 1st 1982

Dear Sirs,

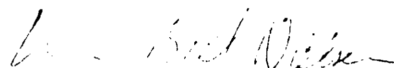
We are using ZEUS (UNIX) on our Zilog System 8000 microcomputers. We are interested in different application packages running under ZEUS.

We would therefore appreciate if you could send us a list of the packages you have available for distribution.

Do you know of any implementation of HASP or JES 2 capable of running under ZEUS?

Also please inform me if your packages are avaiable on ZEUS for format cartridge tapes.

Yours sincerely
SC METRIC A/S



Kim Biel-Nielsen
product manager

**Compiled by Emma Searle, Small Business Microsystems Support, Microsystems Software Unit, SWURCC.**

# Unix machines in transparent networking race

NOW THAT almost everyone has Unix on their favourite machine, the race seems to be on to produce extensions to link lots of them together. And the UK seems to be winning.

At Newcastle University, Brian Randall and his group have developed software to link various Unix systems transparently in local or wide-area networks.

"The software, called the Newcastle Connection, makes the distributed system indistinguishable from a single Unix system," Randall said. "Each user can read or write any file, use any device, execute any command, or inspect any directory, regardless of which system it belongs to.

"It's the most 'fun' thing we've had for a long time. We didn't plan to develop it as a commercial system, but we rapidly found that we couldn't do without it – it's a tremendous convenience."

It tends to make processing more efficient as a command can be executed on any machine in the ring the user does not have to wait until his own machine is free. Also, as many Unix programs contain pseudo-parallel processing, if machines are free they can be executed as *real* parallel processes on different machines.

All this is done transparently, and it will work with any Unix look-alike which is compatible at the system calls level.

The system is being evaluated by Logica, who want to add it to their Xenix systems. Adrian King, divisional manager of the Software Products Group, believes that it is the most advanced system of its kind in the world.

"In the US, Bell Research Labs, Berkeley and other universities are all trying to develop similar systems, but this is better. It has more capabilities, it's the next step. And it's nice that it's British.

"We want to use it internally to connect our own Unix systems. It will be very useful to firms wanting to connect lots of small 16-bit systems.

"At the moment it would only be of value to sophisticated users such as universities and research labs. It will need a few months' work to hide some of the complexities before it becomes a commercial product, but the problems are all identifiable, all solveable."

Newcastle University is collaborating to exploit the product with locally-based Microelectronics Applications Research Institute (MARI).

**Geoff Conrad**

# Unix versions for ICL Per are claimed unacceptabl

ALL available versions of Unix being developed for the Perq computer by the Science and Engineering Research Council (SERC) and ICL are unacceptable in their present form, according to Geoff Manning, director of SERC.

Unix is the operating system favoured for all development work funded by SERC and the ICL distributed Perq, a Motocola 68000 16/32 bit computer, is the favoured machine.

SERC is becoming increasingly frustrated with the problems associated with the project, particularly as it has at least 120 machines awaiting a suitable version of Unix before distribution to Universities and other research centres can begin.

"I'll be very disappointed if it is not running in considerably less than five months," Rob Witty, SERC's Perq team leader, told *Datalink* in March.

Last week researchers from ICL, SERC and Carnegie Mellon University in the US met in London to discuss the future of Unix on the Perq. Carnegie Mellon University is where the 32 bit addressing operating system Accent was developed, and a k of this is used as the basi SERC's version of Unix.

According to Geoff Mann SERC's own version of Un "up and working, but the formance is not everything th required," meaning that th they have Pascal and a For compiler working, the resp times are for too slow for prac use.

The ICL version of Unix, w is also being developed at SE Rutherford Labs, has no po problems but "the Fortran c piler doesn't work yet."

# Unix goes loca

THE Unix operating system ing brought closer to the peop Newcastle University. A soft subsystem developed there a Unix systems to be connected local or wide-area networks.

# Faster tha a speeding Visicalc

VISICALC is under attack a this time from Dynacalc, an tronic spreadsheet from C pusense.

According to Compus director Ted Opyrchalit, Dy is "written in Assemble speed, and will run on any dard 6809 machine under Flex, Uniflex or OS/9."

Southwest Technical Pr is evaluating the product own machines. Software ma Russel Brown is "personally impressed with it. It's ab million times faster than V and has heaps more facilities

"But what I really like is haven't been able to crash and neither has anyone else we have really been trying."

ACT TO LAUNCH OASIS AND UNIX ·



# for the Sirius

computing 16.9.82

Trevor Huggins

T is preparing to launch new operating systems for Sirius 1 16-bit microputer and has unveiled a ech package for the mace as well as a 10 megae Winchester disk drive. on Upton, ACT Sirius' tware manager, told *Coming*: 'We'll be offering the sis 16-bit operating system during the second quarter of 1983 and that will be used as a stepping stone to a release of Unix later in the year.'

Upton went on: 'We see the Oasis system appealing to commercial users of the Sirius while scientific users are expected to take Unix.' ACT is claiming sales of over 3,000 machines since the product — which was designed by ex-Commodore man Chuck Peddle — had its UK launch last autumn.

ACT also launched a 10 megabyte disk drive for the Sirius and an Audio Input Package adds voice input/output to the machine. The package costs £295 and can be used for either voice annotation of text or of programs so that items such as error messages can be spoken rather than displayed.

Other software products in the pipeline include an extended graphics package for the Sirius — entitled Graffix — to make its UK debut in November. ACT is also understood to be on schedule for a November announcement of its networking of the machine using Corvus Systems Omninet network.

● Standards battle, Page 18.

# Unix record amended by chairman

dl 6.9.82

I AM writing to correct some errors of reporting in recent copies of *Datalink*.

● No decision has as yet been taken within AT&T regarding Educational Licenses for System III. That does not imply that there will be none. In fact my information from within AT&T suggests the contrary outcome, in that it is more likely that Educational Licenses will soon be extended to System III. (*Datalink* August 9).

● Your report on the Alvey Committee suggests that there are no database management systems available under Unix. This is an extraordinary statement. Without reference to files I can think of at least seven commercial products, the first of which appeared in 1977. (*Datalink*, August 16).

● AT&T have not yet "thrown off the shackles imposed on it in the US". Their actions are still governed by the 1956 Consent Agreement, under which they are not allowed to be in the computer business. AT&T agreed terms with the Justice Department early this year under which they were released from their Anti-Trust commitment, but these have not yet been ratified by Senate. Indeed, substantial objections have been raised there, and as a result some renegociation is taking place. (*Datalink*, August 16).

**E S Jones, chairman, European Unix User Group.**

*While appreciating Emrys Jones's awareness of the world of Unix, he seems to have forgotten that in each of our stories on these subjects it was made clear that:*

● *"proposals were not firm policy yet. We would like to hear what the academic community thinks of them."*

● *It was not suggested that there was no database management systems available for Unix, just none that are built into the product, ie by Bell.*

● *If anyone seriously believes that AT&T is not going to be allowed to market computers in the near future under Reagan's regime, then perhaps they could explain their reasoning.*

# What are they doing to the SERC Perq?

WHAT are they doing at the Science and Engineering Research Council?

At the beginning of this year, when the Letters page was full of complaints about Perqs at SERC in ...ks and worse, Unix was going to be available in a couple of months. Then in June. Then in August. Then "it's up and working but unacceptable in its present form".

But what has happened to the hundred-odd machines that SERC bought to distribute to university researchers but was waiting until it had Unix before sending out?

At the start of the project, ICL was working with SERC to produce a single version that both could use. Then SERC decided that it wanted to base its version around the Spice kernel developed at Carnegie-Mellon University in the US, so ICL decided to develop a straightforward commercial version. But they carried on working with SERC's Rutherford Appleton Laboratory.

While all this was going on, a Canadian software house quietly brought out an implementation that was snapped up by Three Rivers Corporation, the US company that developed the machine and granted ICL a licence to manufacture it in the first place.

But both British organisations decided to carry on with their own versions.

Last week, when ICL demonstrated its "fairly fragile" implementation at Sicob in Paris (see page 3), it transpired that it had been developed internally and not in collaboration with SERC, whose own version is nowhere to be seen.

Indeed, they were so touchy at SERC last week that they would not let *Datalink* formulate specific questions, let alone cough up a typical "no comment".

# Top level US approval for Micro Focus' Cobol compiler

by Robert Parry

UK SOFTWARE firm Micro Focus has won top level certification for its Level II Cobol compiler. The US government General Services Administration has certified the product at "high level" – the highest of four grades – with no errors. Level II, a microcomputer product, joins compilers from only seven other companies at this level of certification. All seven others are major mainframe suppliers, like IBM, Honeywell and Sperry Univac.

"We really went for GSA listing," says Micro Focus director Stewart Lang. "The micro scene sees lots of changes in hardware, but people are fed up changing their programs all the time."

So far the high level certification is for the Intel 8080 microprocessor implementation of Level II Cobol, but others will follow soon. An 8086 version has reached the second highest grade, and it will be a "straightforward operation" to bring it up to high level says Lang.

Versions to run under Unix will also be pushed for the top grade certification soon.

With the certification of Level II Cobol, the link between microcomputer and mainframe application software has been established, says marketing manager Peter Hewitt. Much of the vast bulk of Cobol application software worldwide conforms to the Ansi 74 standard, as does Level II, and Micro Focus' aim has been to develop portable Cobol compilers to allow transfer of such software to a range of microcomputers.

The GSA high level certification means that US government and federal agencies can now buy micros running level II Cobol. The GSA authorises such purchases, rather as the CCTA does in this country, and demands that the machines run a certified Cobol compiler.

Already US micro manufacturer Cromemco has announced it will offer the Micro Focus Level II Cobol with its 16-bit micros.

# Unix group wants Bell price decision

Computing Aug.

The European Unix User Group has called on Bell Labs to make up its mind about pricing of Unix System III after reports that concessions for universities may be axed.

The commercial cost of a source code licence System III is $43,000 but universities have been able to buy licences for earlier versions of the operating system for $300.

Emerys Jones, chairman of the user group, said: 'Universities would be prepared to pay for the administrative costs of Unix which have been mentioned as about $3,000. But I'm quite sure that none of them could come forward with $43,000.

'We now require that Bell Labs makes up its mind on pricing and comes forward with a clear statement.'

Bell Labs said that it would reach a decision within two months on pricing for Unix System III for academics.

Jones added: 'But if there are no concessions it won't be an unqualified disaster because there are still the Unix lookalikes such as Idris and Coherent.'

There are cheaper and source code for Idris can be bought in unbundled segments.

Jones also suggested that the Computer Board or the Science and Engineering Research Council might be brought in for special negotiations with Bell Labs if university concessions on price of the system were not granted.

# Firm offers Unix for applications

A NEW software house is developing software for the Unix operating system to turn it from a development into an applications environment.

Precision Software is currently developing what its founder, John Tranmer, called an "environment manager" to provide facilities for running the applications software it will be launching in September.

"Unix provides the ideal multiprocessing environment," said Tranmer. "The first thing we had to do was develop an environment manager to take the user away from Unix."

The manager produces a menu controlled system allowing each user access to a private set of utilities and files via passwords. Precision has also made "substantial modifications" to the teleprocessing environment, which, according to Tranmer, offers improvements like easier syste[m] recovery.

Precision will be launching th[e] manager along with a gener[al] ledger package in September with other applications to follow including sales and purchase led[ger], order processing and a tele[x] and viewdata interface.

The company will also be offer[ing] remote software support, in[cluding] access to a central prog[r]am library, with users commun[i]cating with Precision's ow[n] machine via the UUCP commun[i]ciations program, which is one [of] the Unix utilities.

Also, using UUCP, which w[ill] initiate processes on other cpu[s] as well as send ASCII files, [a] message switching system send[ing] mail between users could b[e] set up.

Tranmer said that most of th[e] software has been written in C, the system development lan[guage] used to implement Unix and in Microfocus's CIS Cobol which is used mainly to code dat[a] entry routines.

Precision is developing th[e] software on the UK develope[d] Bleasdale micro, which uses [a] 16-bit Z8000 microprocessor.

Benjamin Woolley

# Precision shows ledgers at Forum

CW 16.8.82

PRECISION Software will be exhibiting two new ledger packages at the European Computer Trade Forum next month.

The Environment Ledger incorporates a full screen editor and menu controlled access to all applications functions, plus Unix type features such as the mail-box and desk directory. This package is aimed at office equipment and catering wholesalers.

The other package, General Ledger, is aimed at the financial world. The system can handle up to 13 balances and maintain them for the current year. It also includes posting facilities, foreign currency conversions, standard reports and trial balances.

Both software packages run on PDP 11's and Vax models and will be available from October.

Precision Software has also been appointed European distributor for the XED word processor and the TXED full screen editor.

Both are products of Computer Methods, the Los Angeles-based software company which specialises in Unix-based word processing systems.

XED is a direct entry word processor. It incorporates a control and command language which may be entered directly or via simplified help menus.

The TXED is a standard ASCII file text editor, providing system developers with a complete range of screen editing facilities.

# Cost of Unix licences for academics to be kept down

A promise to keep the cost of Unix licences for academic users down to a few hundred dollars has laid to rest university fears of a massive price rise.

Frank Riffle, technical licensing manager with AT & T which developed Unix, said rumours that universities would have to pay $43,000 for a Unix System III licence are unfounded. Earlier versions of the operating system have cost academics $300.

'The universities will not have to pay this prohibitive figure,' Riffle promised. 'Education licences will cost something more along the lines of $400, in keeping with our present policy.'

Riffle claimed that AT & T is anxious to hang onto its traditional links with the academic community, in the hope that university users will later become commercial users and pay the full licence fee.

Riffle admitted that $43,000 is rather high for the commercial licence. 'But System III includes both the Programmer's Work Bench and Version 7,' he added.

# Unix version of PDL will aid output

SOFTWARE designers and developers are promised increases in output of up to 30% if they take on and use the Program Design Language (PDL) just brought into this country by WP Computers of Stevenage. And they'll never have to draw flowcharts again.

While PDL has been in use in the US for many years, WP Computers has also announced a brand new rewrite of the system called PDL/81, which works under the Unix operating system.

"The Unix version of PDL is really powerful," said Graham Evans, product manager at WP Computers. "We're hoping to market it with a 16-bit micro system to small software design teams in the UK, but haven't quite decided on the hardware yet.

"The language has two main functions. One is that it can perform very well as a software design tool. The other, and I think more important, aspect of PDL is that it is a method of communication between software designers and coders, the coders and the managers, and so on right down to the end user. PDL caters for everyone."

According to Evans, users of PDL no longer need to concern themselves with the task of physically drawing flowcharts.

"Using PDL requires a certain kind of attitude on the part of the designer," said Evans. "This is because at its highest level it is what you might call a sophisticated word processor, and at its lowest, a software language.

"Once a design has been entered into the PDL system – and this can be in a fairly crude English statement-like form – then the software will automatically be able to generate full documentation of that system, before coding has begun. This has been our biggest selling point."

PDL was developed and is marketed in the US by Cain, Farber and Gordon of Pasadena. Not all price details have been finalised, but the PDL package for a DEC system running RSX-11M costs about £3,300. PDL/81 presently costs £5,000, but Evans said that this could drop depending on demand.

# Atlantic Software goes transatlantic

by Robert Parry

MICROCOMPUTER hardware and software supplier Keen Computers is going transatlantic to focus on its twin enthusiasms Unix and local networks. Atlantic Software, a wholly owned subsidiary of Keen, is to operate on both sides of the Atlantic and in Australia.

Keen specialises in local network and the Unix operating system markets, through its dealerships of Apple and Corvus hardware, and intends that Atlantic Software should act as a clearing house for software products that fit in with these interests.

"The priority is looking for sound Unix products," says Atlantic Software's marketing director Chris Knight. "There's so little of it around". At first the emphasis will be on software for the Unix machines Keen Computers handles – the Onyx micro and Plexus mini – but Knight aims to extend the range of machines he will cater for.

Atlantic will also develop software for Keen's other hardware lines, with products for Apple micros networked through Corvus' Omninet and for the Corvus 16-bit micro, the Concept. The main package for the Concept will initially be a library of subroutines for software houses to use to put together applications packages for end users. The Concept runs under its own operating system, and Knight sees a strong need to promote the writing of applications software for it.

KEEN . . . *"Must concentrate."*

Until now such software activities have gone on inside Keen Computers. With the setting up of Atlantic Software, the new company will take over the work and supply software for Keen. This will include standard CP/M products from Microsoft and MicroPro, UCSD Pascal for Apples, and the range of Unix products from US Interactive Systems for the Onyx and Plexus machines.

The reason for setting up a new company, rather than expanding the software side within Keen Computers, was management controls says Tim Keen. "To make a success out of software you have to concentrate on it," he says. "It doesn't work as part of another operation."

# No lack of Unix DBMS

THERE has been some discussion in recent weeks regarding the fate of AT&T's policy of granting preferential source code licences to academic institutions. My understanding is that no definite decision has as yet been taken on whether these licences will be available for System III or not, but according to informed sources within AT&T it is more than likely that they will; and also that they will be available to universities outside the US.

There is only one fly in the ointment; the present price of $350 will rise considerably to something more in line with what it costs AT&T to issue and monitor these licences. Figures of around $3,000 have been suggested.

While very few universities could afford the full commercial licence at $43,000, I do not believe that $3,000 will cause serious problems for those who are really interested in Unix.

Secondly, I have read the recent articles on the Alvey Committee with some surprise. It has been reported in some quarters that Alvey is recommending a prompt start on a project to write a database system for Unix. The reason given that there is no DBMS available for Unix, implies that either the Alvey Committee have not done their homework properly, or that their chairman has been misreported. Without particularly trying I can think of seven database systems available under Unix, the first of which appeared in 1977.

What also surprises me is that while the Alvey Committee is considering recommending investment in the Unix area they have not at any time consulted the Unix User Group. We are probably better placed than anyone else to report what is currently being done within the Unix community in the UK, and indeed there are now a number of companies in the UK investing in Unix products.

If the Alvey Committee wishes to make an immediate start with something, I suggest that it would be more constructive to co-ordinate these efforts rather than finance yet another database system for Unix.

**E. S. JONES**
**Chairman**
**European UNIX Systems User Group**
**Swansea**

**MORE LETTERS PAGE 16**

neeting of the European Unix Systems Users Group held last week

Computing 16/9/82

# Bell faces attacks on Unix standards

Industry experts are sceptical about Bell Labs' current work on a system of certification to define standards for Unix.

The Unix operating system, a powerful program development tool, is mainly used in universities but as business gets interested standards become an issue.

'We will probably introduce standardisation by writing a program that tests the minimum facilities a system must have to be called Unix,' said Berkley Tague from the Technology Licensing Department of AT&T.

This is how the General Services Administration (GSA) tests Cobol, although it uses a suite of about 300 test programs.

Areas that Bell Labs is seeking to standardise for Unix include networking, multiprocessors, database management and languages.

Applications programs running under Unix are mostly written in C. Tague added that business Unix will need to support Cobol, Basic and Fortran 'and we will have to make decisions about whether we implement full compilers, for example, or just syntax checkers'.

But experts were sceptical about standardising Unix.

In the words of one user, Charles Forsyth from York University, 'once something becomes a standard there is an inbuilt resistance to change'.

'It is a red herring as far as the Unix experts are concerned,' said David Tilbrook of Computer Systems.

'How can you prove a compiler?' asked Cornelia Boldyreff from the microprocessor software unit of South West Universities' Computing Centre.

She hopes that standards would protect users from buying a micro advertised with Unix and finding essential features missing.


## UNIX* and 'C'
### COURSES

As consultants to the training department of the world's largest user of Unix, we offer intensive training workshops in both the Unix Operating System and C Language Programming.

Courses can be organised on a public or in-house basis. Currently we have planned public courses for early 1983 as follows:-

| | |
|---|---|
| Unix Operating System | 17-21 January 1983<br>21-25 March 1983 |
| 'C' Programming Language | 10-14 January 1983<br>14-18 March 1983 |

*Unix is a trademark of Bell Laboratories

## STRUCTURED METHODS
### COURSES

We offer training courses in Structured techniques based on the De Marco approach to structured analysis and the Constantine/Myers/Yourdon design method. The structured analysis and design methods lead into easy structured programming and can complement and interface with existing methods of structured programming (eg; Jackson etc) if required.

Our courses run throughout the year on a public basis, but we can of course offer tailor-made projects to suit.

Our November offerings are:-

| | |
|---|---|
| 1-2 November | – Managing Systems Development |
| 3-4 November | – Reviews, Walkthroughs and Inspections |
| 8-12 November | – Structured Analysis Workshop |
| 15-19 November | – Structured Design/Programming Workshop. |

For more details please contact Andy Ellis

STRUCTURED METHODS (UK) LTD

A SPAN GROUP COMPANY

43-44 GREAT WINDMILL STREET,
LONDON W1V 7PA
TELEPHONE: 01-734 7394 24 HOUR ANSWERPHONE

16.8.82

# …nd Unix to run …n DEC machines

…ERS of Digital Equipment's …' microcomputers will be able …roduce Cobol code at the rate …bout 5,000 lines a day with a …gram generator developed by …company Phoenix Systems.

…hat's the "conservative" esti…e given by the company on the …abilities of the latest version of … System 80 range of Cobol …gram generators.

Additionally, DEC itself has …itten a version of the Unix …erating system to run on the …fessional 350, but apparently … as a test of the machine's …ities.

…hoenix Systems started life … a generator for Prime mini…puters and soon added others … as Texas Instruments and …eywell, venturing into the …o market with a deal to pack… the product for the Tandy …o. Now Phoenix has ex…ded its presence in the micro … by signing up DEC.

DEC liked the look of the …tem, asked us to rewrite it …cifically for its micros, and intends to sell it as a DEC product," said a Phoenix spokesman in the US. "There has been a lot of interest in the System 80."

The DEC version will appear on the Rainbow 100 twin-microprocessor system, the Professional 300 range, the VT180 terminal and the DECmate II word processor.

System 80 products are data dictionary driven and feature a menu selection facility.

Still acting shy about putting a full Unix system onto any of its products, DEC has nevertheless let it be known that it has developed a Unix system that will run on the professional 350 micro.

The motivation seems to have come primarily from a desire to see just how compatible the 350 is with the DEC PDP-11, on which it is based.

One of the DEC staff who worked on the project confirmed the experimental nature of the product, adding that he "wouldn't recommend that anyone tries to use it". **Robin Webster**

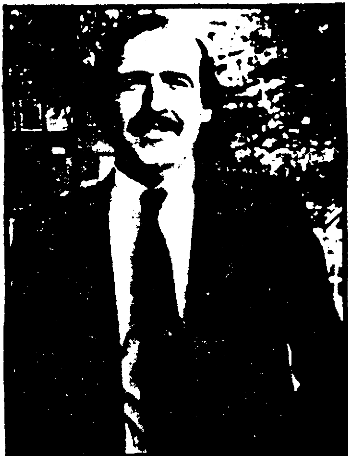
# Relational DBMS for 16-bit micros on Oasis

CW 16/9/82

RELATIONAL database will soon be available for 16-bit microcomputers on the Oasis multi-user operating system.

Systems house SISCO has implemented the Control database developed by Phase One in California on the Oasis system to run on the Executive machine made by the Computer Information Company in the UK, for which it is an OEM. Executive can be either an 8- or 16-bit micro, so the 16-bit implementation will be compatible with the IBM Personal Computer and ACT Sirius 1, both of which support Oasis.

"We have held back from launching the 16-bit version because people have not yet realised the benefits of the second generation of micros," said Patrick O'Brien, sales director of SISCO. "We decided not to be the cavalry and be first over the bridge, but to wait and see what happens."



*O'BRIEN . . . "Not to be the cavalry and first over the bridge."*

SISCO has integrated word processing into Control, and given the package full English language facilities for interrogation using the Access module report writer, to make it suitable for end users.

"Effectively, we have stopped selling word processing and database as separate functions and put the two raw products together," said O'Brien. "It was a case of getting away from bits and bytes, and giving the user a straightforward means of producing personalised letters for about £8,600, including the cost of a top quality printer."

This price puts the SISCO system into the same competition bracket as the Wang and Wordplex word processing systems, but with the advantage of an integral database. Oasis is a more attractive operating system to end users than, for example, the Unix portable system, because of its user-friendliness. It also has good record-locking facilities and, if packaged up with a Winchester disc on the Executive, can support up to five users on an 8-bit machine, according to O'Brien.

"On an 8-bit machine with that number of users, the end user cost is about £5,000," he said.

The combined database and word processing system has already been installed at the Periodical Publishers Association and Charrington Fuel Oils, and at three major banks.

*Software File is compiled by Maggie McLening.*

# …asis destined …or the IBM pc

dl 9/8/82

…IE OASIS multi-user operating …stem, claimed to be more than …t Unix-like by its supporters, …uld feature on a new micro…mputer to be bought in from …atsushita by IBM.

Phase One Systems of Califor…, which developed Oasis, is not …iting around though: it has …eady put the system on the …M Personal Computer and …ns to market it for about …,500 within the next two …nths.

# Wicat joins supermicro race in UK

SOFTWARE Sciences is to distribute a US 68000-based micro in the UK and is looking for software houses and oems to produce software for the product.

The machine is the Wicat 150 micro, based on the Motorola 68000 processor.

Wicat claims 700 are already installed in the US and that forward orders of £7.5 million are already in the bag.

Software Sciences has just set up its own distribution arm – the Wicat 150 is the first product to be picked and the company is on the look-out for more. "We chose the Wicat", said distribution manager Paul Sherry, "because of its competitiveness on price."

The Wicat 150 is a multi-user 16-bit system with 256K of RAM and comes supplied with one 10 megabyte Winchester disk, the Wicat operating system and a choice of one language, wrapped into the price.

Software options include Unix, MSC and a CP/M emulator. The machine will support all the major languages including APL, Cobol, Fortran and C.

Software Sciences has produced a basic range of applications software with Wicat. Packages on offer so far include financial modelling, word processing and business accounting.

The company is also providing a number of hardware options including a graphics screen, RAM up to 1.5 megabytes, a video disk interface and extra Winchester drives plus 5¼ inch floppy backup.

Prices for the Wicat start at £4,995 and a three user system with a 10 megabyte Winchester disk configuration will cost £7,250.

University of Strathclyde
DEPARTMENT OF COMPUTER SCIENCE
**A First Course on UNIX***
6-9 September, 1982
Fee £185
Further information from:
Continuing Education Office
McCance Building
16 Richmond Street
Glasgow G1 1XQ
Tel: 041-552 4400, Ext 2132
*Unix is a trademark of Bell Laboratories

# UK firm runs Unix 7 on 68000 micro

by Robert Parry

BRITISH micro builder Bleasdale Computer Systems is redoubling its attack on the emerging Unix market. A 68000 machine running Unix Version 7 is now up and running, ready for delivery within four weeks, according to managing director Eddie Bleasdale.

"It runs a full implementation of Unix," says Bleasdale, "the real McCoy, not a look-alike version." He claims his is the first 68000-based machine in Europe to run a complete implementation, rather than cut down operating systems based on Unix which only give the user a subset of Unix.

The claim is probably fair enough for European machines, reckons Emrys Jones, chairman of the European Unix Users Group. But he noted that most users did not use all the facilities available within the operating system: "The trouble is they all want different bits, so for a manufacturer it perhaps makes sense to offer a full implementation."

The 68000 machine is a brother to Bleasdale's Z8000 computer running Xenix, the Microsoft derivative of Unix.

It comes with half a megabyte of memory, expandable to 3¼-Mbytes, 10 Mbytes minimum of hard disc with floppy or tape streamer backup, and can support six to eight users comfortably.

*BLEASDALE . . . "£70,000 of orders have been received."*

Bleasdale has £70,000 of orders already, representing four systems.

# 'Desktop mainframe'

"A MAINFRAME on every desk" is the theme behind Hewlett-Packard's worldwide launch this week of a 32-bit desktop computer for engineers and scientists. The company believes it to be the most powerful computer workstation on the market.

The HP 9000 uses the five-chip set announced earlier this year, "super chips" which HP claims pack up to eight times more circuits into the same space as currently available integrated circuits.

With up to three central processors in every unit, the HP 9000 offers Unix and Basic operating systems; Ethernet and HP networking, and Pascal, Fortran and C languages. First deliveries of the single processor HP 9000 series 500 will begin in December, with volume shipments by March 1983.

Announcing the workstation at a worldwide satellite teleconference, executive vice-president Paul Ely called the product a breakthrough in the technical markets, and said it gives the best of both worlds: "Distributed networks of individual computers, as well as the big mainframe benefits of sheer processing power."

# Precision made Unix products

A new software house specialising in Unix-based packages has launched its first products for 16- and 32-bit microcomputers running under this "standard" operating system.

Precision Software is now looking for distributors for its ledger packages and the Environment Manager electronic office system.

Environment Manager includes a diary, electronic mail, a text editor and a calculator. The facilities are called up through menus.

The ledger packages use transaction processing techniques to provide quick recovery from system crashes. Screen and report formats can be created and amended easily. Other features include foreign currency handling and budget control facilities.

The packages cost £950 each.

**Enquiry card: circle 84**

# Developments daily in Unix field

Apart from the graphi add-ons and package Unix seems to be the busie area of software at the m ment. There are ne developments daily, addin to the strong movement make Unix one of the prim software vehicles of th eighties.

Recently two new Uni machines, the US-designe Wicat, and the British-buil Britannia machine runnin Unix lookalike Idris, hav broadened the options sti further.

Unfortunately there ar deficiencies in the operatin system which was originall developed by AT&T's Be Laboratories for research, no commercial operation Because of its desing it is por table, and easy to transfer t other machines (latest i IBM's Series 1), but it lack the end-user facilities whic users expect whatever th well-publicised programmin aids.

Products are beginning make their way onto t market, making Unix an a tractive buy for end-use not just OEMs. Real Ti Systems and Zilog are bo offering Redwood's Unipl word processing softwa linked to their Unix syste RTS's Idris and Zilog's Ze

Precision Software has a nounced a series of busine applications for Unix-bas 32-bit and 16-bit machines, i cluding Nominal Ledger a an office system called t Environment Manager.

"We're trying to mak Unix into a hands-on tool f commercial users," explaine Precision's managing dire tor John Tranmer. "We don mean to build genera purpose tools, but to develo the actual applications, s that end-users hav something which is pleasan to use. Unix should be a pe ple system."

**Enquiry card: circle 117 fo RTS, 118 Zilog, 119 Britanni 120 Precision**

# Modusoft software tool for Unix look-alike

MEASUREMENT Systems (MSL) of Newbury has unveiled a software development tool for the OS9 operating system.

The new product, Modusoft, is a program development tool offering a library of commands and utilities which automatically handles interfaces with hard or floppy disks. This leaves a programmer to concentrate on his program, since he is relieved from screen formatting, data storage and retrieval.

Modusoft incorporates a database manager, designed to run on the powerful OS9 Unix look-alike operating system.

OS9 already has some file handling capabilities, but not a fully fledged database management system.

Using Modusoft on OS9 the user now has multi-tasking and multi-user facilities.

Internally, the database manager contains an indexing system which permits high speed accesses even when using large files. For example, data may be retrieved within 3-5 seconds when using a floppy disk system with 1,000 records, claims MSL.

Modusoft acts like a series of powerful sub-routines and has to be called by any program which needs its facilities. It cannot run by itself.

All the modules are machine code based and can be called from high level languages or assembler programs, without any degradation in speed. They can be provided as disk or PROM based.

Dave Davies, managing director of MSL, said: "The OS9 is infinitely superior to its major competition, CP/M, but will become more popular when it has the same amount of software back-up."

dl 16.8.82

# Unix deal

ROOT Computers has concluded a deal with UniSoft of Berkeley, California, to port the Unix System 3 operating system to micros based on the Motorola 68000 chip.

# DEC to support Unix

DEC "intends to get into Unix business in earnest," said a source at Digital Equipment Corp, in an exclusive interview.

This follows DEC's June announcement of VNX, a "combination of the VMS operating system from DEC and Unix-like enhancements for the VAX." The trend within DEC toward supporting Unix, the popular operating system from Bell Labs, has been marked by several milestones:

- Formation of Unix special interest groups within DECUS, the DEC users' group;
- Formation of the Unix Eng neering group.
- Announcement that DEC wi provide free of charge Unix de vice drivers for DEC peri pherals;
- VNX, announced in June which makes VMS look lik Unix to VNX users.

These milestones, coupled wit news that DEC "now offers similar set of Unix facilities fo DECSystem-20 hardware," indi cates official DEC support fo Unix soon.

9/8/82

# Newcastle Connection spreads the Unix net

Robert Parry

), not a football team. Unix ...ited, otherwise known as the ...wcastle Connection, promises ...nsparent user access to distri...ted systems running under Unix ...erating systems.

A software subsystem developed ... the computing laboratory of ...ewcastle University, it is incor...rated into a set of standard Unix ... Unix look-alike systems and ...ows them to be connected to...her into local or wide area net...rks.

...xploitation of the Newcastle ...nnection is to be co-ordinated ... MARI, the Microelectronics ...plications Research Institute. ...RI is part-owned by the Uni...ity, Newcastle Polytechnic, ... CAP. London-based software ...se Logica is expected to be the ... to supply it commercially to ...users, as an extra feature for ...Unix implementation Xenix. ...ca will install an evaluation ...m next month.

... distributed system using ...castle Connection software is ...tinguishable from a conven...l single system as far as the ... is concerned. Inter-processor ...munication is hidden from the ..., who can access devices on the ...ork – within the normal ... password control constraints ... though they were part of his ...system.

...o particular network hardware ...ecified. As no modification of ...Unix kernel or applications programs is needed, the set-up can be used with any Unix-like system compatible with the original Bell Laboratories Unix at system call level. Different Unix implementations can be mixed in a Unix United system.

"That's the beauty of it," says MARI's general manager Bob Cooper. "Provided we're talking about genuine Unix or Unix look-alike systems we can talk across anything. It really doesn't matter what network is used, as we are sitting on top of the communications."

The original implementation at the University of Newcastle is running on a Cambridge Ring, but only because that happens to be the network used there.

Logica will install a system for evaluation in London, to see what needs to be done to turn it into a real commercial product. The system will probably be fitted early in August, says software products group development manager Adrian King, the delay being due to a move of offices.

Cooper reckons the Newcastle Connection will be available in the marketplace in about three months' time. After Xenix, he expects applications to other Unix-like operating systems to appear, possibly led by one for Idris, the Unix look-alike marketed by Real Time Systems based in Newcastle.

A case of another Newcastle connection?

# CCTA shuns Unix

**The CCTA recently announced that it will give higher priority to microcomputers offering CP/M and BOS operating systems rather than Unix. The Central Computer and Telecommunications Agency (CCTA) is a government procurement body which hopes to establish between 6-12 approved suppliers of microcomputers for government departments.**

**This recent decision has created a lot of controversy and disbelief since Unix is rapidly becoming a standard. Unix is a multipurpose multi-user operating system designed to run on the new generation of powerful 16 bit microcomputers as well as on a wide range of minicomputers such as PDP-11s and VAXs. It is receiving a lot of backing from the hardware and software producing areas of the industry with the result that there is a wide selection of machines, databases and applications available on Unix. A government department, the Science Research Council has recently installed a Unix system and Britains Universities are also backing Unix.**

**Logica, who have received a lot government aid are now selling Xenix a version of Unix under license from Microsoft and have submitted a proposal to the CCTA.**

**There are only two machines available, from Corvus and Wycatt, which allow CP/M, BOS and Unix capabilities together. While Bleasdale Systems, a British firm who have recently received £50,000 in government loans, is leading the European market in delivering and installing Unix systems, and would not be eligible to compete for the CCTA contracts. Emris Jones, chairman of the European Unix Users Group told *Micro Forecast*, "this is a very strange decision and I would like to know the reasons for it". It seems likely that the CCTA like the Alvey committee, has not done its homework on Unix and will eventually have to change its decision through commercial pressure.**

## Unix moves

Computing 16/9/82

Pressure is building up for commercial users to abandon manufacturers' operating systems and turn to Unix. Adrian King of Logica (above) told last week's Unix conference in Leeds of an emulation for VAX users, due for launch next month. Full report of the meeting, **Page 15.** Meanwhile News Analysis looks at the Government's decision not to authorise Unix, **Page 18**

# ...EC micro to ...et C compiler ...ut not Unix yet

...E DIGITAL Equipment Pro...sional 350 microsystem is to ... a C language compiler, but the ...ision will not necessarily lead ...Unix being put on the same ...tem.

...ast week DEC confirmed that ...ad been experimenting with ...ting the Unix system onto the ..., but "had no immediate plans ...ell the system commercially". ...owever, it seems the C com...r developed during the project ...ets DEC marketing standards.

# APL68000 now running under Unix

THE FIRST APL interpreter running under Unix on a 68000-based micro has been announced by Codata Systems in California.

APL68000 is a full implementation of IBM's APL.SV, which was released in 1972. It was written by Philip van Cleave for The Computer Co (TCC).

"The 68000 is the first microprocessor with the horsepower to do justice to APL," van Cleave said. "It's really a 32-bit processor with a 16-bit data bus; with its 32-bit registers it looks like an IBM 370."

APL operators treat arrays as basic data structures, so an APL machine must be capable of moving around large blocks of data. The 68000's 32-bit internal architecture means that it can address up to 16 Mbytes of memory and swing large blocks of data around.

DL 6/9/82

# Microsoft to rule 16-bit o/s market?

US SOFTWARE company Microsoft is seeking to dominate the 16-bit operating system market with two new versions of its MS DOS product and a rewrite of its Xenix multi-user system around Bell Lab's commercially targeted Unix System III.

The first new MS DOS release, to be made this month, will be version 2.0 and this will be supplied to current MS DOS version 1.0 oem customers – including IBM, Hitachi, NEC, Wang, and Digital Equipment – free of charge.

The second addition will be a multi-tasking version of MS DOS, although this is not expected until early next year.

And paving the way for Microsoft's new Xenix in the UK will be Logica, which is shortly to announce a version of Xenix incorporating some of the major features of Unix System III.

Although MS DOS version 2.0 will support all version 1.0 system calls and programs, the new product will not necessarily come with a CP/M look-alike user interface, as was the case with version 1.0.

Rather, users will be presented with a menu-selection interface that makes far better use of the file and operating system structures that lay behind the original "false" CP/M front-end.

"What we're trying to do is build a software bridge between MS DOS and Xenix," a US spokesman for Microsoft told *Datalink*. "Version 2.0 is really quite an update. If an oem wants to take version 2.0 and put a CP/M front-end on it, that's up to him, but we've decided to give the user the ability to create his own interface, or what we like to call a visual shell."

Other enhancements include: upward compatibility with the Xenix operating system: the inclusion of pipe and filter facilities typically only found in Unix or Unix look-alikes; and a cursor pointing feature which allows commands displayed on a terminal screen to be executed by simply positioning the cursor alongside the required word.

Xenix compatibility has been achieved by introducing new system calls into MS DOS which give it the same tree structured file directory that Xenix employs.

Extensive HELP facilities have been included in MS DOS version 2.0 and these are invoked when user types in a "?"

In MS DOS pipelines and filters could be used to sort directories or format and title document files.

Logica's involvement in the popularisation of Xenix will come in the form of a Digital Equipment PDP-11 version of Xenix which will appear soon. This will include the Source Code Control System (SCCS) and a remote job entry connection to mainframes facility taken straight out of Bell Lab's Unix System III.

Microsoft *is* working on a System III Xenix, but intends to do the complete upgrade job in one go with a release date set for the end of the year.

It appears the Logica will switch over to Microsoft's upgraded Xenix when it appears, but additional plans are being laid.

"Logica will offer an implementation derived from System III on the Vax and other machines, and we will be providing full support of the system," said a Logica spokesman.

**Robin Webster**

## Unix gets another defender

*From John McEvoy*

I must take brief issue with Jane Bird over some rema[rks] on Unix in her article on National Computer Con[fer]ence (*Computing*, July 8).

Record locking facili[ties] have existed for some time [in a] number of Unix variants, [not] least TSC's Uniflex marke[ted] in the UK by South W[est] Technical Products.

I simply cannot accept [the] hopeless lack of user frien[dly] facilities with Unix'.

Unix itself and most of [its] variants offer an extrem[ely] powerful command interp[re]ter, 'shell', and a perfe[ctly] adequate macro build[ing] facility.

The combination of the t[wo] means that operating pr[oce]dures can be custom[ised] rapidly so as to provide [the] user with systems which [...] perfectly acceptably.

The naive user need n[ot] know that they are talkin[g to] an operating system. [The] experienced user will de[light] in its combination of sim[plic]ity, flexibility and power.

I would venture to sugg[est it] may be lack of skill in pac[kag]ing applications rather [than] deficiency in the opera[ting] system which has led Bi[rd to] her conclusions.

*John McEvoy, director, C[om]puter House (Turnkey), Newbridge Street, New[castle] Upon Tyne.*

## System III in the UK-hallelujah!

ROOT Computers has clinched its first orders for Unix System III in the UK – the new version of the Bell operating system that was released last December.

It has also released two software packages for System III: a spooler suite and a source code control system.

According to director Michael Kinton, "We are the only people in the UK to have System III up and running."

One order is from Microlease, who want eight systems running on DEC PDP 11s to control a mailing list for the Billy Graham Organisation.

## Getting down to the root of the matter

FURTHER to your article about Unix running on Vax (*Datalink*, September 13), we should like to draw your attention to the fact that Root Computers has been supplying and supporting Unix System III in native mode on Vax since the first Root installation in June 1982.

**D J Saunderson, Root Computers, London EC1.**

Root director, David Saunderson, sets the record strai[ght]

# ogica's Unity is he first Unix for DEC's supermini

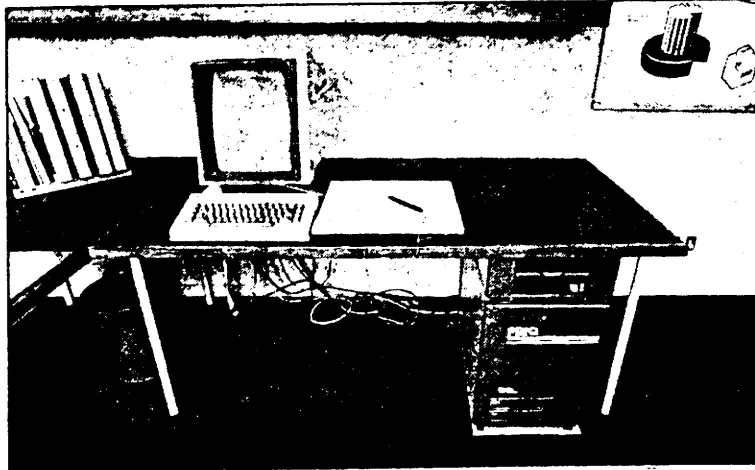
DL 13/9/82

. European Unix Users ip meetings are becoming e and more a springboard for luct launches. Not only did week's meeting attract the t of AT & T, with its own uncements, but also a hand- UK users saved up releases e event.

example, Logica took the e to at least verbally launch a Unix look-alike system for igital Equipment Vax 32-bit omputer, and another for L Perq.

ither was available for scru- but just as soon as Logica ts Vax machine delivered in ple of weeks then Unity, ted from Human Computing rces in Canada, should be ble.

spokesman on the Logica claimed that this was the 32-bit Unix available on a l machine, with the excep- of the Berkeley Unix, which on a Vax, but is only Unix on VII, rather than the latest m III.



Logica still looks to be ahead on introducing a Perq Unix

At first Unity will only run as a program under Digital's VMS operating system, but later a faster native mode version will be released.

But the Perq Unix, like everybody else's, isn't really ready yet. The spokesman added: "At one time we thought it would be quite a while, but things have speeded up and we now expect to be running Unix on the Perq by October." If he manages that schedule, he could still beat both ICL and the Science and Engineering Research Council.

One company, Root Computers, used the conference both to claim a victory over Logica and to announce a couple of products of its own. Root has just landed a contract with John Mowlem, a large publicly owned construction company, to supply Unix-based machines for internal development of software.

This was won in competition with Logica, but seemed to be due primarily to the fact that Logica is not yet selling a System III version of the look-alike system, Xenix.

Logica has one almost ready to go, but it isn't fully System III compatible, since it is waiting for the authors of Xenix, US-based software house Microsoft, to come up with a full System III type version.

At the same time Root has also decided to offer the Unix Source Code Control System, for use on System III Unix. It was a bit unclear as to why this warranted an announcement, since it is part and parcel of Unix, but some suppliers, Root claimed, do not supply it. Root also announced a spooler package for a variety of printers.

Perkin Elmer was busy telling everyone what its adverts have been telling us all for the past week or so: that it has a special offer on Unix systems based around its 3210 machine, a 32-bit minicomputer.

The offer is a 15% discount for everyone qualifying for an educational licence for Unix. Perkin Elmer also supplies it with the Source Code Control System, but currently offers only the Version VII Unix.

**Peter White**

# ax gets Unix features

DL 9.8.82

TAL Equipment is taking it sees as the "most signi- features" of the Unix oper- system and putting them on ax.

ile other companies already complete Unix systems on achine, DEC has decided it be highly selective in its e of Unix facilities and let demand dictate product de- ment.

oducts are being developed under the VNX banner, and two have just been released on the market: the Vax-11 C compiler and the DEC/CMS (Code Management System) package. Both run only under the VMS operating system.

"The Vax-11 C compiler is an extended version of the C programming language system already available on the Vax," said a DEC UK spokeswoman. "The DEC/CMS system is a tool for the management of the software design process.

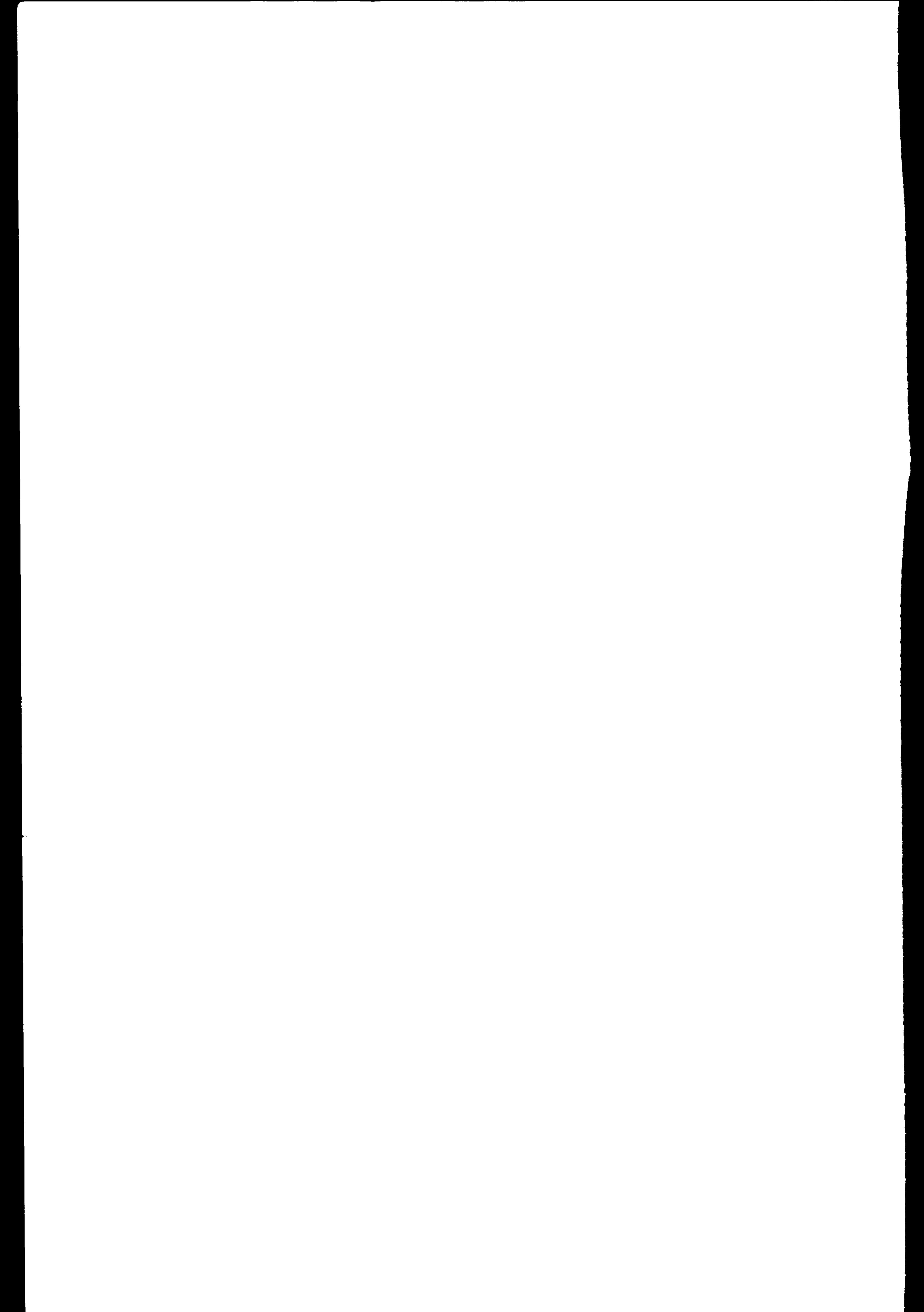
"They are just two of what will be a number of Unix facilities that will be made available to Vax VMS users from the VNX development programme in the US. This is not to say that we are putting the Unix system on the Vax – it's more a case of providing the significant Unix features to meet the demands of Vax customers."

With the Vax-11 C compiler, it will be possible to write and execute C language programs under VMS.

The DEC/CMS system is essentially a version of the standard Unix Source Code Control System (SCCS) which, in the Unix environment, controls and records the creation and revision of programs.

Both products will be available in the UK by early next year.

**Robin Webster**



The Secretary
**European Unix User Group**
Owles Hall
Buntingford, Herts.
SG9 9PL.
Tel: Royston (0763) 73039.