AUSTRALIAN UNIX USERS GROUP NEWSLETTER

LAST USERGROUP MEETING

Approximately seventy people attended the last User Group meeting held on the campus of the University of New South Wales on September 12th last. See reports supplied by the speaker this issue.

NEW NEWSLETTER EDITOR

Since Ian Johnstone is leaving in November to become chief of our overseas office, (he is going to work at Bell) I should like to introduce the new Newsletter editor. I am Peter Ivanov, from the Dept of Computer Science at UNSW, and all further items of interest for publication in the Newsletter should be sent to me at the address on the last page of this issue.

I cannot believe, having seen the vast numbers of Newsletters being produced here, that its readership can be so incredibly silent. Dont you people know how to write? Please, please, please send us some material to publish or I will keep filling future issues up with the same sort of junk Ianj used to put in (God help us all).

Peteri.

NEWSLETTER CONTRIBUTIONS

In order to meet demand and defary costs which are now substantial a charge is to be made for the Newsletter. The cost for a years subscription (6 issues) is $12 Aust. Back issues (5 so far) may be obtained for $10 Aust. All future issues will be sent only to those who have paid !!

A word of warning, since the last Unix User meeting, at which the topic of money was first raised, only five subsriptions have been received. A readership of 5 is not very inspiring.

A continuing lack of interest may result in the editor losing interest also, so avoid the rush!!!!!!! Return the form on the last page of this newsletter with your subscription as soon as possible.

WHAT MORE CAN I SAY

Quote for the month:

> "Happiness is not knowing what
> the bits in the error register mean!"

Ian Johnstone
AGSM
PO Box 1
Kensington 2033
AUSTRALIA

(02) 662-3752

TELEPHONE
345 1844

TELEGRAMS
UNIMELB PARKVILLE

University of Melbourne

DEPARTMENT OF COMPUTER SCIENCE

Parkville, Victoria 3052

17th August, 1979.

-INTERNATIONAL UNIX USERS MEETING-

The Australian UNIX Users Group is planning to hold an International UNIX Users Meeting in Melbourne to co-incide with the Eighth World Computer Congress and Exhibition (formerly IFIP '80).

At this stage, we are seeking initial responses from those interested in attending such a meeting, with a view to estimating the likely number of attendees and preferred dates.

The IFIP Congress will be held in Melbourne from October 14-17, 1980, following a session in Tokyo (October 6-9).

Could you please copy this notice and circulate it as widely as possible within your known circle of UNIX converts.

We plan to distribute a formal notice and invitation to attend to all UNIX licence holders by December 1, 1979; consequently we would appreciate responses to this initial "flier" as soon as possible in order that we may finalize the dates and start organization in earnest.

Regards,
Ken J. McDonell.

Dr. Ken J. McDonell,
Department of Computer Science,
University of Melbourne,
Parkville, Victoria, 3052,
A U S T R A L I A.

Please complete and return to the above address as soon as possible:

Name: _____

Affiliation: _____

I shall/shall not be planning to attend the International UNIX Users Meeting.

I prefer the following date(s):

Sunday October 12 - Monday October 13

Saturday October 18 - Sunday October 19

Monday October 20 - Tuesday October 21

Other : (please specify).

(Please strike out those dates on which you would prefer the meeting not to be held).

AGENDA UNIX USER GROUP MEETING 12th September 1979

9:30	Morning tea
10.00	Introduction
10.05	Who What Where When etc...
10.30	John Lions (UNSW) UNIX Manuals
10.40	Piers Lauder (Sydney); Bob Kummerfield (Sydney) Local UNIX Networking
11.10	Ross Nealon (Wollongong) Graphics Application Package
11.30	Dennis Ritchie (Bell) Geraniums and their Application to the Control of Student Riots
12.00	Lunch
14.00	Chris Rowles (Sydney) Who Needs LSI-UNIX when MINI-UNIX will do?
14.15	Greg Rose (UNSW) Local Site Report
14.30	General Business i. Next Meeting ii. Newsletter
14.50	GRIPES hardware/software
15.05	Terminal Control
15.30	Afternoon Tea
16.00	Site Visit - UNSW

PROPOSED POLICY STATEMENT TO BE
DISCUSSED AT THE NEXT UNIX USER MEETING.
PLEASE FEED BACK COMMENTS ASAP.

UNIX Manuals

J. Lions
University of New South Wales

In the past, the University of New South Wales Computer Science Department has acted as a principal agent for the reproduction and distribution of UNIX manuals, and we are happy to continue in this role. However some new considerations will enter into future activity: Level Seven UNIX comes with a greatly increased volume of supporting documentation; some of the previous level of secrecy can now be relaxed; and changes to the book bounty act will affect printing costs after January 1, 1980 (in effect, we may lose a 25% subsidy). Facing us are the problems of what to print, when and how many?

First, we assume that there will be a finite demand for sets of the complete manual, but these will not be for everybody (especially if they cost of the order of $40-$50 per set).

Second, we propose to continue our present UNIX Beginners Manual which is intended primarily to serve the needs of first and second year Computer Science at the University of New South Wales. The format and content of this manual are by no means final, and suggestions will be welcomed for improving it or making it more generally useful.*

Also proposed is a UNIX Text Processing Manual to contain the complete word processing documentation, plus enough introductory material to make it self-contained as a starter manual.

Under present consideration is a proposal for a UNIX Companion, which, under current thinking, will supplement the UNIX Beginners' Manual for our third year computer science students. It is suggested that this should contain Section 2 and Section 3 and 4 of the UPM; reference and tutorial material on C; the UNIX Assembler Manual; the original Ritchie/Thompson paper; the YACC reference manual, etc.

There are many other possibilities which could be made and might generate sufficient support to be viable. Please make your views known soon.

---

* It currently contains "A Guide to Using UNIX", "Edit: a Tutorial", introductory and reference material for Pascal, and a subset of Section 1 of UPM.

Dear Ian,

Here is a brief re-run of my talk about local networking at the recent AUUG meeting.

Our current system can be described as a user controlled network, and as such has been simple to develop. At the root of the system is the story of one elegant program originally known as "rogin", and its evolution into its present form, known as "log". "Rogin" stood for "remote login", the idea being that if two tty lines in two different systems were connected together, and the line on the remote system had a "getty" controlling it, then all that was necessary to get access to the remote system was a program that copied your input to the remote line, and simultaneously copied the output from the remote line to your terminal. "Rogin" did this by forking and, essentially, executing two "cat"s with the flow going in opposite directions. "Rogin" also had another aspect which enabled a rudimentary form of file transfer. A quit code enabled the user to break out of "login" mode and invoke file transfer mode. "Rogin" had two aliases, "send", and "receive", which were invoked in the remote system in response to the user requesting file transfer. If the user typed "get file", "rogin" would send "send file<CR>" to the remote system, and then copy anything up to an "end-of-message" sequence into the local file.

I might mention that at some point "rogin" got renamed to "1" and later to "log" - its current name.

There were two problems with "log", one was that it was prone to being swapped out, which while being annoying in "login" mode, was disastrous in file transfer mode, as the file received was not necessarily identical to the one sent, which brought out the second problem - there was no protocol for ensuring reliable file transfer. The second problem got solved first with a simple protocol that has remained unchanged to date, and appears to guarantee file transfer under all conditions.

The first problem has been solved by adding an extra system call "connect". This connects two tty lines together such that input on one line is immediately added to the output queue of the other line at interrupt time. "Login" mode is now simple - the "connect" call is used with three parameters, first file descriptor, second file descriptor, and the ascii code which, when detected on the first file, will "break" the connection. "Log" then takes no further part in the proceedings until woken up by the quit code being typed. There are some further improvements to the "connect" call made recently which allow a super user to intercede on another user's terminal and control it remotely.

"Log" allowed an informal network to grow up around the campus of New South Wales University where ever two or more systems were connected together. Preferably there were two lines between each system, with one "getty" on each at opposite ends.

Now for the multiplexed network which arose as follows. AGSM and Basser decided to share the costs of a leased line connecting their two systems, and initially communication was in one direction only from Basser to AGSM where a "getty" was patiently waiting. This, while fine for users from Basser wishing to use the AGSM system, was not much help for users from AGSM. Various quite dreadful proposals were made, (e.g. two "getty"s in some sort of dialogue deciding whose attempt to use the line should succeed). But the final solution was inspired by a Unix driver, known for some obscure reason as "sys.c". This

is an "indirect" driver implementing the pseudo device "/dev/tty". Why not have a pseudo device which turns one "tty" line into many virtual ones? And indeed it works, making use of the code already driving the "connect" call, but its effects are otherwise transparent to the system. This is known as the "mx" driver and provides any number of virtual channels on any number of real "tty" lines. The "mx" driver achieves its multiplexed channels by virtue of a simple protocol consisting of a header containing a byte count, followed by "count" data bytes. The header is 4 bytes long, so programs using 1 byte writes are maximally inefficient, however the observed degradation in file transfer time has only been about 7%.

At Basser we now have two virtual lines to AGSM, each with a getty on it at AGSM, and another virtual line with a getty waiting for AGSM users "logging" to Basser. There are other virtual lines which will be used by network mailing daemons, background file transfer daemons, etc.

A quick tour through the network can happen like this:- a user logs in to the Basser system and types "log agsm", the "log" program polls the various lines whose names start with "agsm" until it finds one that isn't being used, and replies:-

```
*tty
basser -> agsm ('^a'=quit)
```

the user is now "connected" to the AGSM system. Having logged in at AGSM, let's move on ...

```
"log elec"
*tty
agsm -> elec ('^e'=quit)
```

gets us connected to the UNSW Electrical Engineering Computing Facility, and so on.

Problems are that on multiple "log"s the user must remember to log out in the correct order, and file transfer cannot jump systems.

Piers Lauder
Basser Dept of Computer Science
Sydney University

Graphics Assistance Package.

The GAF system consists of a set of user-callable subroutines, and one or more interpreters. A user program calls the subroutines, which process data supplied through the argument list, and produce a sequence of commands and arguments to these commands, which are structured in such a way as to be graphics output device independant. The interpreter then processes this file of commands, producing output (a picture) for a specific device. A program calling GAF subroutines may be used as a filter coupled directly to the interpreter via a pipe, so that the system may be used to produce graphical output interactivly.

The subroutines are grouped into seven broad catagories:-

1) low level utilities.
   These consist of the basic primitives necessary to produce graphical output. Some of these are clear (clear the screen) dmove (move the drawing 'pen' without drawing a line, a dark-move), vdraw (move the pen, but draw as you go). Other primitives are built upon these, such as the routine ldraw (draw a line by darkmove then vector draw).

2) system definition.
   These routines allow the user to select and specify his or her world, to select windows through which to view the frame, co-ordinate systems (and hence scaling factors), and make use of auto scaling when the best co-ordinate system is not known.

3) graphs.
   Routines here draw axes, with variable numbers and levels of tic marks, draw a solid or broken line through a series of points, draw a histogram, graph with error bars, annotate axes and graphs. There is one routine that combines all of these features to enable the user to produce a very reasonable labeled graph with very little effort.

4) grids.
   Subroutines allow drawing of log or linear scaled grids over any area of the frame.

5) text system.
   A software character generator allows drawing of either software or hardware characters on the frame. The user specifies the size of the software characters as a percentage of the frame size; fractional percentages are allowed (i.e.- .007 is 7/10 ths of a percent.) Multiple fonts are allowed, and strings of characters may be drawn either horizontally or vertically.

6) symbol definition.
   One of GAF's strong points is the ability to define a sub-picture, or symbol. This is equivalent to subroutines in

programming languages. Display of symbols allows re-scaling the symbol (varying its size) and hence distortion of the symbol is easily achieved. The symbol may be viewed through a user selected window, and hence a zoom into a symbol may be effected by repeatedly changing the window size.

7) grid systems.
The grid system allows the definition of the frame as a two-dimensional 'array' or matrix of grid sectors, and several subroutines to allow plotting of symbols in the sectors by specifying the co-ordinates of the sectors (the indices). Routines allow definition of a vector of symbols, and plotting the whole vector over the grid. The percentage space occupied by the symbol may also be specified.

The system has as yet no three-dimensional stuff in it, but has been designed to accomodate the 3d primitives easilly. The package also lacks a contouring routine, and one to shade polygons, due to lack of time on the author's part.

The motivations behind GAP are:-

1) to produce an easy to use, user friendly system for producing impressive and useful graphical output by the novice and experienced user alike,

2) to make this system as concise and simple, yet powerful, as possible. The primitives, therefore, must be well chosen or the package will grow without bounds trying to accomodate 'features' that do not meld together.

The package was designed with the GD3 and PLOT systems in mind, though none of the code from these systems has been used. The GD3 system is currently in use at CERN, in Geneva on cyber series machines. The PLOT system (together with the PUREJOY and NCAR and some other systems) at the University of Wollongong, on a UNIVAL-1106.

I do however, acknowledge the invaluable Work in design and method and styling of the packages GD3 and PLOT, to A. Yule, R. Miller, A. Jeavons, C. Piney, P. Ward of GD3, and P. Castle of PLOT. I must apologise for my failure to credit these people in my brief talk on GAP at the UNIX users meeting.

Ross Nealon,
Department of Computing Science,
University of Wollongong.

TELEPHONE 370 0111
TELEX—UNIVQLD AA40315
TELEGRAMS—BRISBANE UNIVERSITY

YOUR REFERENCE:

OUR REFERENCE:

University of Queensland

ELECTRICAL ENGINEERING
DEPARTMENT

ST. LUCIA, BRISBANE
AUSTRALIA, 4067

29 June, 1979

Mr. C. Rowles
Department of Chemical Engineering
University of Sydney
SYDNEY. NSW. 2000

Dear Mr. Rowles

In this department we are configuring an LSI-11 for computer control and data acquisition in our microwave lab. The hardware will consist of an LSI-11/2, 32K RAM, BDV-11 bootstrap/ROM, DLV-11-J, VDU, GPIB interface, ADC's, DAC's, and a serial line to our PDP-11/40 which is running under UNIX. As you can see from the lack of secondary storage we intend to use the LSI-11 as a satellite to the 11/40.

I understand from Clary Harridge, our Computer Technologist, that you are currently enhancing the MINI-UNIX operating system with interprocessor communication capability.

Could you please advise us:

1. Would the software you are developing be suitable for our system?
2. On what basis could we obtain your software? (We have applied for a MINI-UNIX licence.)
3. How much work would be involved in adapting your software to our system? Would you like us to do some of the development work?
4. Could the software reside in the 16K ROM in the BDV-11 for copying into RAM on booting the system? Or would it be preferable to always down-line load the software from the 11/40?
5. We expect our hardware in September. When will your software be ready?

Thank you for your assistance.

Yours sincerely

Michael A Husband

(M.A. HUSBAND)
Lecturer in
Electrical Engineering (Computer Engineering)

CASE WESTERN RESERVE UNIVERSITY
SCHOOL OF MEDICINE

DEPARTMENT OF ANESTHESIOLOGY
UNIVERSITY HOSPITALS
2065 ADELBERT ROAD
CLEVELAND, OHIO 44106
TELEPHONE: (216) 791-7300

Mr. Chris Rowles
Chemical Engineering
University of Sydney
Sydney, AUSTRALIA

July 3, 1979

Dear Mr. Rowles:

I am a user of Mini-Unix software on a PDP 11/20 computer. I was referred to you by Ian Johnstone at the recent Unix conference as someone who had solved many of the problems associated with the original Mini-Unix distribution. I've experienced many problems in having the Mini-Unix software run reliably and I would greatly appreciate learning of the bugs which you found. (My problems are characterized by frequent "crashes" of the system).

Yours truly,

Carl Thompson
Systems Analyst

Anesthesia Department
University Hospitals
2065 Adelbert Road
Cleveland, OH 44106
USA

TELEPHONE:

or: 692 2455

The University of Sydney

THE DEPARTMENT OF CHEMICAL ENGINEERING

N.S.W. 2006

Block Structured Character Devices

Miniunix had some difficulty in handling PC-11 paper tape system on my 11/20. Punching was fine, reading 300 characters was fine, beyond that the unibus hung. Clearly a need existed to solve the PC-11's problem. The solution gave rise to some interesting questions.

The PC-11 driver consists of two parts (a read, reader and a writer, punch). On pcopen a check was made for multiple read requests (by more than one process) with returns a device busy error. No such check has been implemented for "write". Hence it is possible to have many processes concurrently writing to the PC-11, resulting in some very R rated tapes. Most people should be aware of this design feature, and some effort to correct this state of affairs could be made. The obvious solution is to separate the pc structure into two parts. The PC structure was:

```
struct clist {
        int  cc;
        int  cf;
        int  cl;
           };

struct pc11 {
        int pcstate;
        struct clist pcin;
        struct clist pcout;
        } pc11;
```

This was modified to:

```
struct pc11 {
        int pcpstate;
        int pcrstate;
        struct clist pcin;
        struct clist pcout;
        } pc11;
```

To separate the drivers the easiest method is to make two separate flags pcpstate and pcrstate testing and setting them independently of each other. This ensures that the reader and punch, in fact, operate as independent units. Note this approach requires least recoding of the driver. The pcopen must test the writing state, for multiple users, then set the pcp state to WRITING.

This fix allows only a single writer on the PC-11. But this did not cause the original 11/20 bus hang.

The problem arose due to the passc (spl6) function operating in close proximity to a 50Hz clock (br6) and a 150 Hz paper tape reader (br4). A design fault in the 11/20 caused a race condition that was aggravated by the PC driver synchronising with the line clock. The 11/20 was slow anyway. I looked to see what could be done to 'improve' the operation of passc ( ). I hit upon the notion: why use passc ( ) in this device at all?

Passc ( ) collected the incoming characters in response to the PC-11, and stuffed them into a buffer (eventually) as dictated by a programmed read request on the PC-11 (via iomove type function). If you could grab a bigger (à la DP driver) then process the read request to get minimum (or lesser) of the number of characters to be read or a buffer (256) then it is possible to fill the buffer directly from the PC-11 interrupt service routine. Furthermore, on EOT, the block can be marked as a short fill and iomoved back to the user process. This has enormous savings in character processing, and it allowed the PC-11 to work on the 11/20.

The result of this is a hybrid driver, block input, character output. Failing to have sufficient buffer space to allow the block output mode, the PC-11 development stopped as it was. However, with an 11/60 and unix (and a buffer surplus) I am considering a block output mode for both the PC-11 and LP-11. The overhead in this mode of operation is one buffer and one buffer structure (+ pointer) while the device is running, as the buffer is released upon device closure. This change in operational mode does not have any affect upon the mechanism used to invoke I/O from the program environment. It merely reduces the processing load upon the well defined character devices. (A spin off is that this structure allows a nicer connection to a line control front-end, used to process high speed character devices logically attached to a larger Unix host).

Further mods:

The PC-11 is a much loved and entrenched device. Back up copies of user programs help nervous users, and it allows input from microprocessor data acquisition systems, so the driver was developed to new heights. You guessed--a raw driver and an ASCII driver.

With the separation (logically) of the units, a minor device modifier (bitØ) was used to specify the reader (=Ø) or the punch (=1). Two device names were used to access the reader/punch dev/pr and dev/pp.

The raw drivers are invoked by bit 1 in the minor device number (called dev/rpr and dev rpp). A raw reader is an 8 bit binary reader. A raw punch is an 8 bit binary punch.

An ASCII reader stores only valid printable ASCII characters (plus tab,/n, /r,/b) rejecting all others and performs a parity strip.

An ASCII punch is really a binary punch that converts /n to /r/n. Well who can think of anything else that it should do?

Super PC is available as an ASCII source or, you guessed it, paper-tape.

C.D. Rowles
Dept. Chemical Engineering
University of Sydney

TELEPHONE:

or: 692 2455

The University of Sydney

THE DEPARTMENT OF CHEMICAL ENGINEERING

N.S.W. 2006

September, 1979

Release Note: Mini Unix

Product Description:

A Unix-like operating system that will run in any (all) PDP-11 CPU(s).

Mini Unix is an unmapped version of Unix. It supports up to 4 users and 13 processes (in standard version). The system takes 12K words of core leaving 16K words of user space.

More processes can be supported (up to 30 have been used in a version running in a student environment at this department).

The penalties for large (4) numbers of users and multiple processes are:

(a) The system must grow beyond 12K. A very difficult thing to arrange (in terms of the logistics of what to compile next in order to end the exercise with a running system). The version in use at S.U. is 14K words long, which really represents about the upper limit, especially if a C-compiler is desired.

(b) Every extra process requires more swap space. At present the swap size is fixed at

$$SWSIZE = NPROC * (user\ 2/4) + 2 * 512\ bytes$$

and this space must be made available on the file system. This is a serious drawback for floppy disc based systems.

(c) Swapping has a response time penalty. The main objection to swapping is that there is only one process in core at any one time, hence swapping starts with each new process. For the smaller members of the PDP-11 family this is a very time consuming effort.

Enhancements over Bell's Distribution.

Several features have been included in Miniunix:

(a) The mean time between system crashes has been increased from 20 minutes (1978) to 4 months (1979).

(b) The system will run in LSI-11 (PDP-11/03) processors.

(c) Modified shell that does not give rise to multiple shells hanging around after a user logs off. In the original version it was possible to log in a new user at a terminal, leaving the previous user's shell in tact. In fact, three or four terminal sessions later, the process table resembled Shelly beach (littered to and

fro with live, but unusable shells). The current mods kill shells on log out.

(d) Reorganised source file structure. In the original system the parameter files (eg. param.h, conf.h, user,h) lived as double entities in two different directions (dump still does not understand about links). Reconfiguring the system resulted in two edits (ideally at the same patch level) or two versions of the parameter file being used by various parts of the operating system. To overcome this a new directory /USR/SYS/PARAMETERS was generated to hold all .h files.

(e) Brief "how to" notes are included for those brave souls who find a 12K system a little too small. Recompilation of the support environment (cat, ed, fc, cc, etc.) takes 1 to 2 days depending upon the machine used. The run files for the sources on these disks must be modified before they can be used. Some indication is given as to how this must be modified.

(f) Additional user support is likely to become available as more people use MINI-UNIX.

Performance:

Single user miniunix is very acceptable. It tends to be slow (especially on a PDP 11/05) and some patience is a prime user prerequisite for its use.

Multiuser miniunix is a sorrier story. Editing files is reasonable but compilation and execution is a little slow (mainly due to swapping). It is not recommended for the PDP 11/05. On a PDP 11/20 with 4 users response times were on par with the main CDC 72/26A system during an average day time loading. Terminal response for editing is much better than the Cyber, and benchmarks using a compute bound fortran program (with floating point calculations) indicate that execution of the Cyber was 300 times faster than the 11/20, but the elapsed terminal response was only twice as fast. This was not a rigorous test by any means.

The LSI-11 gives a response almost equal to the 11/20, when it is fitted with a KEV-11 option, hence multiuser 11/03 systems are practical if a fast disk is available for swapping. Floppy based systems will be less than speedy.

Future Development:

By the time this report is published, miniunix will be floppy based for 1 or 2 floppy drive types. Portability to additional drives will be dependent upon new floppy disk drives being available. At present I have a RX.Ø1 driver (from U.N.S.W.) and I will develop a driver for A.E.D. 6200 floppy. As I do not have access to other floppy disks further drivers will depend upon the end user of the system. If any one has any floppy disk drivers, they can be sent for inclusion in future Miniunix distributions.

Distribution:

At present miniunix sources lies on RKØ7 packs on a PDP-11/60. I have direct access to RKØ5 disk drives, and the distribution comes in the form of three 3/4 filled RKØ5. (binary desk, Level 6 manual disk and Level 6 source disk).

The RKØ5 can be rolled onto 600 ft. mag tape easily and 2400 ft. tapes with some bother, but this will change with time (as 2400 ft. becoming more readily available). At this stage floppy disk distribution is not possible. The AED 6200 version may become possible, but that represents the limit in the forseeable future. (nb. I do not own the floppy drive and hence a disinterested and indifferent attitude to any particular type of floppy drive. I did hear that DEC are about to release a new double density double sided floppy (IMb capacity) soon).

The outcome of this is that distribution is on mag tape and is not bootable in 11/03's (most 1103's not having mag tapes). The mag tape does include an operating 11/03 system (LSMX) and a real processor system (MX).

Why do it anyway?

LSMX was developed after Bell decided not to release LSI Unix (LSX). It was undertaken to give a unix-like environment for 11/03's within the Faculty of Engineering at the University of Sydney. This is its sole claim to fame, and should not be considered in any other light.

With the acquisition of an 11/60 the need to develop miniunix further has lost some urgency. I intend to use it as the vehicle to drive an 11/20 sattelite to a Unix based 11/60 system. Unix will use the 11/20 to drive the character interrupt peripherals (cull to cyber, line printer, paper tape reader/punch and a high speed home grown line network within the Faculty of Engineering). As such, I expect the 11/20 to have no disk of its own, and will use a link to provide to operating system and communications with the 11/60.

Copyright and boring things:

Mini Unix will only be released to users owning a current Mini Unix license. At this stage I will also expect to recover "out of pocket" costs in providing distribution media.

C.D. Rowles
Dept. Chemical Engineering
University of Sydney

Implementing UNIX on a PDP-11/34
or
Wnat does the `F' in "RK05-F" really stand for ?

Dave Horsfall
Computing Services Unit (CSU)
University of NSW

This article is relates some of the author's experiences in implementing UNIX on a PDP-11/34. My efforts were not quite as straightforward as they could have been, since all my previous experience has been with 11/40's. This article will point out some of the major differences between the two models that affect system implementation, and gives some advice to would-be purchasers of 11/34's intending to run UNIX.

Most difficulties appeared when trying to transport an 11/40 system to the 11/34. The first difficulty that cropped up (apart from the lack of a console; I'll come to that later) is the lack of a stack limit register. This was actually the result of a modification to UNIX to utilize the register should it exist, to prevent the kernel stack from smashing the per-user area. Also, the code to handle bus traps had a few nasty side effects. This showed up when using `m40.s' as a basis for the low core program. This meant that all code referring to the stack limit register had to be `conditionalised'. This affects `mch.s' and `once.c'. If the FPU floating point unit is installed (as it usually is; at least on the 11/34's I have seen) then `mch.s' must be changed to support it. The CSU system generation procedures assume a generalised `mch.s' program with appropriate features conditionally assembled in; `mkconf' will generate a file to be prefixed to `mch.s' containing the definitions such as "FPU = 1" etc. `Mkconf' has been extended to recognize keywords such as "stacklimit", "fpu" etc and generate the appropriate prefix file. Since we generate most of the UNIX systems on campus for PDP's of various configurations, 'mkconf' is a great help.

The second difficulty is the lack of a conventional console. Given that UNIX refers to the switch register quite a lot during various stages of booting and running, this is indeed quite a problem. Instead of a conventional switch register and display, there is an arrangement of little buttons and a seven-segment display, vaguely reminiscent of a pocket calculator. There is no switch register as such; you have to button in a number (which appears in the display; it's quite fun just watching it) then press a button to actually load the damned thing, whereupon an LED come on to indicate that the display is indeed displaying the switch register (as opposed to displaying something else e.g the last value examined). This lack of a display such as `ADDR/DATA' also means that you can't tell what the system is doing - if it is doing anything at all.

The boot procedure is quite funny (funny queer; not funny ha-ha; although it does have its hilarious aspects). While holding down the `CNTRL' button, one presses the `HALT' button, then the `BOOT' button. The ROM console emulator program then comes alive on the console terminal. If you are booting from say `RK0' you then type in `DK0' (which must be in upper case, although you can say just `DK'). It would be a good idea at this point if the switch register contained neither `0' nor `173030' (for obvious reasons; none of this "Load Addr 773110; Start" business). The value I normally use is

plain `l'. At least there is a button to clear the switch register. Given that the ROM expects upper case, and that UNIX prefers you to talk in lower case, it is quite easy for problems to occur here.

The subtitle of this article refers to the fact that the two 11/34 systems I have installed both have an RK05-J (the normal one) and an RK05-F (double density; equivalent logically to two drives but on one cartridge). DEC must be flogging lots of these. The implications of this is that there is but one removable volume, and UNIX (for all its reliability - sorry Ian) requires two for effective system backups. The second volume does not have to be a disk; it can be a mag-tape drive. Given that you can copy half of the `F' on to the `J', followed by the other half; the question that naturally arises is "How the hell do you back up the `J' if it is not being used as pure scratch ?". How indeed ! The only technique is to bring up a stand-alone utility such as `RKDF' and when the `F' is nicely backed up, you copy the `J' on to one half of it, copy this in turn to another `J' then restore said half of the `F'. This also introduces another problem; that of recovering from file system loss. Should a file be spread over both halves of the `F', it will naturally appear on two `J's, and the only way to recover it (for the ordinary mortal user) is to restore the whole damned lot in a manner analogous to the backup procedure. It is also possible to treat the RK as a tape and use `dump/restor' on it. However, you still need some sort of emergency backup system just in case you can't even use `restor'. Ken Higgs of Water Research Laboratory can tell a few stories about this. In other words, one removable drive (with or without a fixed drive) is not enough. You need either a second removable drive or a mag-tape drive as well. Life may not be meant to be easy, but there is no point in making it hard either.

There is another consequence of the RK05-F in that UNIX regards it as two platters, and will therefore try to initiate a seek on one of them while the other is busy. Needless to say, this doesn't quite work. We modified our RK driver to implement the concept of `concatenated volumes', in which several contiguous drives may be treated as one enormous drive with one scheduling queue. Should this then be specified as "optimized" (with the funny rotated blocks) the driver does the right thing and plants block #1 in the middle of the whole virtual disk (this calculation is done automatically by the driver; it knows how big each individual drive is and hence knows where the middle of the whole lot is). This does not work too well however on multiple physical drives, but it is better than nothing.

All in all, transporting UNIX from a PDP-11/40 to an 11/34 is not quite a trivial exercise, and it is arguable whether the problems encountered stem from the idiosyncracies of the 11/34 or from UNIX itself ...

EDITORS NOTE:

Some people have more problems than others. Chris Maltby and Ian Johnstone have not had similar experiences !!

DCIEM
1133 Sheppard Ave. W.
PO Box 2000
Downsview, Ont.
M3M 3B9
(416) 633-4240 ext.300

Ian Johnstone
Australian Graduate School of Management
Univ. New South Wales
Kensington NSW 2033
Australia

Dear Ian ,

The dust has now settled and I am able to send you this pile of "junk mail" arising out of the conference in Toronto.

The hand-written bundle of notes is a merging of notes taken by myself and Greg Hill at the last conference in Santa Monica. The first three pages are a result of a site visit to Interactive before the conference started. Dennis Mumaugh also took notes and they are supposed to appear in a newsletter sometime.

The typeset document is the "scribe" notes from the Toronto conference. It is most likely to be distributed via the newsletter or a special mailing from Mel, but who knows when. Feel free to pass it around your user group right away.

The last bundle is listings of the C preprocessor as it comes in the standard I/O package (the typesetter C compiler or Version 7 C less enum and a few other small things); the preprocessor as modified at DCIEM to include code from NSW distribution cc.c that permitted extended if(n)def expressions; and a diff listing of the two. This change will allow you to compile NSW code under version 7 C.

Hope to see you again at future conferences.

Your truly,

Sandra Wright

Sandra Wright

University of Essex

Department of Electrical
Engineering Science
Wivenhoe Park
Colchester CO4 3SQ

NEW TEL. NO.
(0206) 862286

Tel: Colchester 44144 (STD Code 020 6)
Telegraphic address: University Colchester
Telex: 98440 (UNILIB COLCHSTR)

Ian Johnstone
agsm
UNSW

4 September 1979

Dear Ian,

The fifth UK Unix newsletter has just been sent out. I asked Decus (who do the printing and distribution - free!) to mail overseas copies by air, but don't know whether they did or not, and anyway don't know how long they will take to get to Australia. There may be some cheap rate for printed matter. Anyway, do let me know if you don't receive a copy soon. You may want to put some of the material in your own newsletter, and if so I can supply you with better copies if you need them - ours is reduced and double-sided as it is about 150 pages long. I will try to produce it quarterly from now on if there is enough material.

Nobody here has version 7 yet, though several have ordered it. It will probably become the UK standard, with a stripped-down version made for 11/34's and /40's.

All the best,

Bruce

D. B. Anderson

L. E. R. S.

Laboratoires d'Études et de Recherches Synthelabo

58, rue de la Glacière - 75621 PARIS Cedex 13

Société Anonyme au Capital de 2.700.000 F. R. C. Paris B 69 1294 Téléphone : 589 99 29

September 14, 1979

Ian Johnstone
AGSM
University of New South Wales
P.O. Box 1,
KENSINGTON
N.S.W.
Australia 2033

Dear Dr Johnstone,

I was interested to read your letter to UK Unix news. (A brief note on UNIX system performance). We are running a similar if slightly smaller 11/70 system in a commercial environment (pharmaceutical research). Although we have not done the sort of measures that you have done we are of course interested in improving system performance. One or two things we have done recently or have heard about may be of interest to you.

We have three DZ11s (4 soon) and have recently purchased a KMC 11A auxiliary processor and written software to use it to drive the DZ11s (KMC 11A assembler included). This software is as yet not fully tested but appears to work. At a later date we may be able to distribute it.

Secondly we have heard of a way of effectively speeding up the 11/70 unibus by making it operate in 'burst mode'. This modification is being done by Systime in England in connection with their 11/70 based system and SMD controller. We are hoping to borrow some modified 11/70 boards from them in the near future to see if the modifications are compatible with our systems Industries SMD controllers.

Please let me know if I can be of any further assistance in those directions.

Yours sincerely,

Ian R. Perry

Ian R. Perry
Chef du Service Informatique

THE UNIVERSITY OF NEW SOUTH WALES

ROYAL MILITARY COLLEGE • DUNTROON • A.C.T. • 2600
TELEPHONE CANBERRA XXXXXX
66 3741

PLEASE QUOTE

FACULTY OF MILITARY STUDIES
DEPARTMENT OF MATHEMATICS
J. C. BURNS
PROFESSOR AND HEAD OF DEPARTMENT

19th September, 1979

Mr I.L. Johnstone
AGSM
University of New South Wales
P.O. Box 1
KENSINGTON. N.S.W. 2033.

Dear Ian,

As discussed at the last UNIX User's Group meeting, I am sending along three mag tapes for you to fill with goodies.

First of all, will you please arrange for our installation to be registered under the UNSW UNIX licence. The machine serial number is 907, and it will be located in the Department of Computer Science, UNSW, Faculty of Military Studies, in building A82 at the Royal Military College, Duntroon. A.C.T. 2600.

We will definitely want the following software:

1. UNIX Level 7,
2. PWB,
3. Berkeley PASCAL,
4. Vrije PASCAL,
5. York Modula and BCPL,
6. Kent BCPL,
7. ALGOL 68,
8. Sussex POP2,
9. SPITBOL,
10. Toronto Graphics Packages,
11. Hebrew University Tape.

The Computer Centre has purchased DEC licences for this machine for FORTRAN IV, FORTRAN IV plus, and BASIC-11 to operate under IAS and so we are eligible to receive any equivalent UNIX packages you can give us under the UNSW licence. We may have some use for these, particularly in transferring our current software to UNIX, so would appreciate receiving them.

You also suggested that a "Wish List" might be the best way to request software about which we have no detailed knowledge, but which you may be able to supply. I list the categories in order of importance to us:

1. Digital circuit, CAD packages such as simulation packages, draughting aids, and printed circuit board and backplane wiring preparation aids;

.../2

2. Graphics packages, especially for GT-40's;

3. Packages to allow code to be prepared using UNIX facilities which may be down-line loaded to a PDP 11/10 or an LSI-11 (mini-UNIX?), or to compile UNIX on a PDP 10 for down-line loading to a PDP 11;

4. Any system code or packages which you have implemented to improve system security under level 6;

5. Generalized Cross-assemblers and/or linkers for micro-processors;

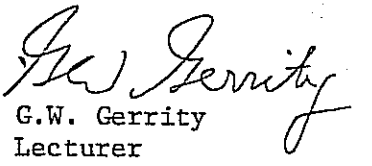
6. Compiler - Compilers;

7. Any other compilers or interpreters not mentioned above for high-level languages (eg LISP, other Pascal).

For your information, the system has the following configuration:

1. PDP 11/45 with 124k words core memory and FPP,
2. 1 - RP04 disc drive,
3. 3 - RP05 disc drives,
4. 2 - 9-track, 1 - 7-track TU10 mag-tape drives,
5. 1 - DH11 serial line controller with 12 - 20mA ports and 4 - E1A ports,
6. KW11-P real-time clock,
7. 2 - GT-42 graphics display systems,
8. 1 - FACIT 250 char/sec Dot-Matrix printer
9. 1 - FACIT Paper tape reader and punch,
10. 1 - 12 in. digital incremental drum plotter,
11. 4 - hard-copy terminals.

Yours sincerely,



G.W. Gerrity
Lecturer
Department of Computer Science

RESEARCH SCHOOL OF PHYSICAL SCIENCES
THE AUSTRALIAN NATIONAL UNIVERSITY

TELEPHONE: CANBERRA 49 5111
TELEGRAMS: NATUNIV, CANBERRA

BOX 4, P.O.
CANBERRA
A.C.T. 2600

PLEASE QUOTE REF.

Ian Johnston
Australian Graduate School of Management,
University of New South Wales,
P.O.Box 1,
Kensington,
N.S.W. 2033

21-Sep-79

Dear Ian Johnston,

Your name and address were given to me by Malcolm Smith of Duntroon,in reply to an enquiry I made concerning the UNIX Users Society Program Tapes.

We have already been provided with the UNIX operating system and documentation by Bell Laboratories under an agreement with the Western Electric Company,and we are now interested in the Users Society tapes.

If you could advise me as to the availability and cost of these Program Tapes I would be very grateful.

Hoping to hear from you soon.

Yours sincerely,

Julie Dalco
School Computer Unit

Nijmegen, nov. 20, 1978.

Dear prof. Ferentz,

As a homage to all people concerned with the security of PDP-11 operating systems, we would like to present this little UNIX program to the readers of ;Login:

```
/ to be assembled and linked to a 407-type loadmodule
/ H.J. Thomassen, E.G. Keizer, Nijmegen, the Netherlands, oct. '78
/ Warning: chase away all users and 'sync' before starting this
/
        mov     $162400,sp
        tst     (sp)            / get stack page down to 160000 sharp
        sys     17.;160000      / break; push up program space to 160000
        br      1f
/ Now you have full 32 Kw allocated; no dirty tricks so far.
        . = 226^.
1:      mov     pc,sp
        jsr     pc,-(pc)
```

alternative!

The first line may be system dependent, although this variable is seldom changed from the original one. The value is 160000 + (100 * SIZCR) with all numbers in octal.

This program will crash any 11/45, 11/50 and 11/55 system. The clue is that these CPU's have a bug in the microcode of the 'spl' instruction. It is left as a nice exercise to the reader to see how this relates to the above program! The microcode does not allow any interrupts to come in in between an spl and the next instruction. It does not even check the setting of the console 'halt' switch thus you'll have to use 'INIT' to stop the above program.

We are certainly not the only ones that have found this bug. It has even been published [Popek J., in: Proc. of the fifth symp. on operating systems principles; 19 - 21 nov. '75; the University of Texas at Austin; ACM-SIGOPS, Vol. 9, nr. 4, 1975, pg 105]. Although this publication has been before the introduction of the 11/55, DEC had not fixed the bug. We must assume that this was for reasons of compatibility between the models.

Since a user starting this program is identified easily from a system dump, we do not expect much mis-use resulting from publication. It stays a good example of how hardware may undermine the good intentions of software designers.

Sincerely,

Hendrik-Jan Thomassen.

Parametrization of software distributions

Johan W. Stevenson

Wiskundig Seminarium, Vrije Universiteit, Amsterdam.

A consequence of the lack of support for UNIX is, that these systems diverge quite rapidly. Moreover, because most UNIX systems are used at universities and other educational or research institutions, UNIX users tend to repair bugs, improve programs and develop new software themselves. Because a central organization for collecting and distributing bug reports and new software is missing, UNIX systems will continue to diverge. One of the great disadvantages is that the software exchange will become more difficult.

Last year we tried to distribute some of our home made programs, notably our Pascal system. Because the software was not stabilized when we started distributing, we had to make several distribution kits. Each time it took several days to collect all files, to produce a 'run' file and a 'READ_ME' file, and to test it. The results were discouraging. Each time we overlooked some little, but necessary, changes we made to the standard software. Each time the 'READ_ME' file was too long and too cryptic to be understood except by very experienced UNIX users. Each time we failed to note some of the problems concerned with installation on machines without hardware floating point unit or without an 11/45 type memory management unit.

To save you the same troubles, we shall give you some hints. First of all, what does a good distribution kit look like?

- Installation must work on the standard UNIX system as distributed by Bell Labs. Non-standard software needed for installation, like archivers, compilers, assemblers and loaders, must be included. The distribution medium must not give problems for standard systems (do not use special tape or disk formats).
- If special hardware is needed, this must be clearly indicated. Examples: 11/45 type memory management because some processes need more than 64k bytes logical address space, special terminals with cursor addressing or graphic capabilities.
- All sources must be included. Binaries alone are useless.
- All places likely to be changed to fit the installers installation must be clearly indicated. These places include all occurrences of pathnames, cursor addressing routines and other terminal dependant features, and places where a trap to the floating point interpreter must be included.
- Besides the normal documentation like tutorials, reference manuals and 'man' files, there must be an installation guide, preferably on a file called 'READ_ME'. This file must explain which changes to the standard UNIX software are necessary, which hardware is needed and where possibly changes are to be made.
- There must be a thoroughly tested shell command file, preferably called 'run', compiling and installing all distributed programs.

Adapt to standard UNIX.

It is the distributors task to make the distribution kit suited for the original UNIX system, the common ancestor of source and target installation, because only he knows of the changes made to his system. Because most programs are not written to be distributed, a lot of additional effort is required by the distributor. It appears to be extremely difficult to identify all places making use of local features, simply because one gets used to these features.

However, there are two powerful aids. The first is to maintain a list of local features to be checked when making distributions. This list may also be useful for new users of your system. An example of such a list is given in the appendix. The second and most valuable aid is a copy of original UNIX to test your distribution kit. A final test may be to let an arbitrary UNIX user install it on an original UNIX system.

To illustrate that it is really very hard to find all trouble spots, we give you two examples.

In the early days we noticed that the kernel saves the floating point registers on a process switch without changing the floating point status to double mode. This means that only 32 bits are saved and restored, whereas there may be valuable information in the least significant 32 bits. This information is lost. Most C programs do not have any problems, because they do not change the floating point status. The results of Pascal programs however were unpredictable. There is an easy fix, but we forgot to include it in the distribution kit.

Occasionally the library routine 'fcvt' did wild stores when converting small floating point numbers, because it did not check for array bounds. Even when such bugs are included in the check list, you might overlook that the routine 'fcvt' is used.

Ease the installers job.

It is the installers responsibility to adapt the distribution to his system. But the distributor must ease the installers job by pinpointing the installation dependent code. Each hour spent by the distributor may save several hours each time the software is installed. It asks for some imagination to predict which code must be changed to fit another installation. Some examples are: pathnames and cursor addressing options.

Many pathnames are used more than once. For instance the pathname of a library used by some compiler you are distributing will occur in the compiler source as well as in the 'run' file. Installation would be a lot easier if each pathname has to be changed only once and if all pathnames are found in the same file.

For C programs these conditions can easily be satisfied by using the '#include' facility and by concentrating all pathnames in a single file, say 'local.h'. The same pathnames can be used in command files as follows:

- give your old 'run' file the name 'run.c'.
- insert at the front of 'run.c'
  ```
  #include "local.h"
  ```
- replace all explicit pathnames in 'run.c' by identifiers defined in 'local.h'.
- add the following 'run' file:
  ```
  cc -P run.c
  sh run.i
  rm run.i
  ```

The flag '-P' causes only the C-preprocessor to be run. The file with suffix '.i' contains the output of the preprocessor. Note that strings, surrounded by double quotes, are treated correctly by the shell and that C-type comments are allowed in 'local.h', because they are removed by the preprocessor.

The preprocessor is not only useful for shell command files. Several times we used it for assembly programs as well. A nice example is the UNIX kernel. All assembly routines for the kernel are concentrated in one assembly file. However, there are two different versions. The file 'm45.s' is for systems with separate address spaces and, as option, a hardware floating point unit. Systems without these features use 'm40.s'. This caused several problems.

First, the 11/60 has hardware floating point, but no separate address spaces, so there had to come a third version 'm60.s'. Second, there are some other differences between PDP-11 machines, notably the presence of a UNIBUS map device on 11/70 machines. Code was included in the kernel to test the global variable 'cputype' at run time. Third, while 70% of these two or three files could be identical, it was clear that they suffered from the multiple copy update problem. At several places the code was different without doing a different job.

All these problems may be resolved by merging these files into the file 'm.c'. The preprocessor should account for the differences. Translation of the file 'm.c' can be done as follows:

```
cc -P m.c
as m.i
mv a.out m.o
```

Some aspects of the preprocessor may cause problems when applying it to non-C files:

- only C-type identifiers may be defined and replaced.
- only C-type strings are recognized; this is relevant, because strings have to be copied literal.
- only C-type comments are recognized; within comments there is no text replacement.

Another disadvantage is, that too many control lines like #ifdef and #define result in hard to understand files, especially if the #ifdef's are nested.

But when used moderately it is valuable tool making software exchange easier. Moreover, the preprocessor is available on every UNIX installation. Making a distribution kit is not harder, because the 'READ_ME' file can be much shorter, and installation is a lot easier.

EXAMPLE: Check list.

- do not use Yale shell commands like 'cd', 'set' and 'newbin'.
- avoid the use of separate address spaces where possible.
- especially avoid the use of separate address spaces together with floating point instructions, because the floating point interpreter needs mapped address spaces.
- do not use local library routines or include them in your distribution.
- the following library routines may cause problems elsewhere, because we repaired some bugs:

  pipe: register r2 was not saved before use.

  ecvt/fcvt: they did some wild stores for small reals.

  atof: leading '+' was not recognized, not even in the exponent.

  wait: used nargs() before, so could not be used together with separate address spaces.

- the following changes to the kernel may affect portability:

  the times system call has been changed in accordance with the diff-listing ditributed by Bell.

  saving of floating point registers is changed so it works for processes using floating mode.

- do not use the -o flags of 'cc', 'as' and 'ld'.
- avoid empty replacement text for '#define', because this may cause the next line to be skipped.
- Because some tables are increased always check your distribution on a original UNIX system.

EXAMPLE: The files 'local.h' and 'run.c' for Pascal.

The parameters to be tuned by the installer of Pascal are found on the file 'local.h', which looks like:

```
/* comment out if you do not have separate I and D */
#define SEPARATE_ID     1

/* comment out if you do not have hardware floating point */
#define HARDWARE_FP     1

/* path names */
#define AS_PATH         "/usr/bin/as"           /* assembler */
#define LD_PATH         "/usr/bin/ld"           /* loader */
#define PC_PATH         "/usr/bin/pc"           /* analog of cc */
#define PEM_PATH        "/usr/ovl/pc:pem"       /* compiler proper */
#define LIST_PATH       "/usr/ovl/pc:list"      /* list errors */
#define OPT_PATH        "/usr/ovl/pc:opt"       /* Em1 optimizer */
#define PDP_PATH        "/usr/ovl/pc:pdp"       /* translate to 'as' */
#define RT0_PATH        "/usr/lib/pc:rt0.o"     /* run time startoff */
#define FRT0_PATH       "/usr/lib/pc:frt0.o"    /* idem, with fptrap */
#define BSS_PATH        "/usr/lib/pc:bss.o"     /* end of bss */
#define EMLIB_PATH      "/usr/lib/pc:emlib.a"   /* Em1 library */
#define PRLIB_PATH      "/usr/lib/pc:prlib.a"   /* Pascal library */

/* do not change the five zero's at the end */
#define TMP_PATH        "/usr/tmp/ptmp00000"
```

- -

The Pascal run file makes use of several run files in subdirectories. Only the file 'run.c' is preprocessed, the others are plain command files. Here follows the file 'run.c':

```
#include "local.h"

: indicate if 'pem' may use both address spaces, I and D
: if not, then only moderate sized programs may be compiled
#define PEMFLAG 0
#ifdef SEPARATE_ID
#  define PEMFLAG 1
#endif

: boot pem by 'pc pem.e', not by 'pc pem.p'
#define PEMBOOT 1

#define FFLAG           /* some spaces as replacement*/
#ifndef HARDWARE_FP
#  define FFLAG "-f"
#endif

cc -O -s pc.c; mv a.out PC_PATH

cc -s list.c; mv a.out LIST_PATH

chdir emlib; sh run
mv pc:rt0.o RT0_PATH
#ifndef HARDWARE_FP
mv pc:frt0.o FRT0_PATH
#endif
mv pc:bss.o BSS_PATH
mv emlib.a EMLIB_PATH
chdir ..

chdir prlib; sh run                ; mv prlib.a PRLIB_PATH; chdir ..
chdir opt  ; sh run FFLAG          ; mv opt OPT_PATH      ; chdir ..
chdir pdp  ; sh run FFLAG          ; mv pdp PDP_PATH      ; chdir ..
chdir pem  ; sh run PEMBOOT PEMFLAG; mv pem PEM_PATH      ; chdir ..
```

A short evaluation of the Fortran IV+ compiler as distributed via Commercial Union Leasing Corp. N.Y.

by Nijmegen

We got the compiler in may 1977, payed US$ 3000.- (commercial price $ 6000.-). Conditions: should have a DEC Fortran IV+ licence (since it really is only an adaptation of this compiler to UNIX), no source, no maintenance, update for UNIX V7 promised at 'a reasonable fee'.

The compiler supports split I&D space, overlays, and it has been extended with an interface to all UNIX system calls ('chapter II'). As with DEC's Fortran IV+, the run time system requires a PDP-11 with the large floating point instruction set (no PDP-11/40, although a software simulator is available). In order to utilise split I&D space and overlays together, the UNIX-"diff" changes as announced in the US ;Login newsletter of November 1976 are required. They are not distributed with the compiler.

Examples of how to make calls from and to C modules are given, although they make quite a kludgy impression. The latter might even be said about the package as a whole. The same is true for connection to Macro-11 modules using the Harvard-macro11 software package as distributed on the first Harvard-UNIX distribution. This Harvard software was built partially by the same people that converted the Fortran IV+ for Commercial Union. The compiler requires a special linker (distributed with the system but the same as Harvard's) and librarian.

The operational system consumes close to 600 blocks in the /bin, /etc and /lib directories (cannot be put elsewhere). The compiler is 3-pass, loading the passes over and again for each (sub-)routine in your program. Each pass is a module of about 85 blocks (stripped), thus people already close to thrashing should stay away from it.

The code generated looks of the same quality as DEC's code, thus good. Heavy calculations run faster than they do in an equivalent C program, which is probably due to more efficient calling conventions. There is an option to get the generated assembler source code out. There is some magic involved in using the compiler: When hunting after a bug, just re-compiling helps quite often.

The documentation is all machine readable, although some texts are in roff instead of nroff format. It consists of a full set of manual pages, including some for chapter V, and an entire new chapter to serve as a 'shadow' for chapter II. It also includes the above mentioned examples for linking to C, some stories, a run file etc. No language reference manual is provided: You do have a DEC licence with their manuals anyhow, although that will never match exactly.

The address for more information is:
Commercial Union Leasing Corp,
645 Madison Avenue, New York, N.Y., 10022.

A short evaluation of the Algol68S compiler as distributed by the University of Manitoba, Canada (prof. P.King).

by Nijmegen

We got the compiler in april 1978, payed US$ 250.- (commercial price is higher). Conditions: no source, no maintenance (but comments are welcomed). For a description of Algol68S vs Algol68 see: Hibbard, P.G., A sublanguage of Algol68, SIGPLAN notices 12, 5 (may 1977).

The compiler consists (you don't believe this) of an RSX11 load module! A special loader gets executed, does a core-break (II) and a large stack-bump, overwrites the now contiguous 32Kw user memory area with a copy of that load module and hops right into the middle of it. The loader has set up provisions to catch (signal (II)) the EMT-traps of the RSX-module and to convert them to UNIX-system calls.

The original compiler was developed at Carnegie-Mellon Univ, Pittsburg Pa. The UNIX adaptation as we got it was designed for UNIX-V5, but worked under V6 after only a slight fix to the above mentioned stack bump. Source for the UNIX-loader and terminal interface is included, as are the RSX-object modules, so under RSX are some modifications possible.

The compiler needs a large run-time system; user programs are limited to 4Kb of generated code (200-400 lines in Algol68S). The ratio of 4Kb vs. 200-400 lines of source means that the generated code is quite compact. There are no provisions for linking to C or to other languages. Nor the compiler nor the run-time system supports shared texts, split spaces etc. Among the fancy things it does support are parallel processing and event-variables.

The documentation consists of an (out of date) CMU-Algol68S manual. Some things described in there, e.g. traces and other debugging aids, don't work. Other problems are with I/O (like []STRING) and with recursive MODE definitions, but there is nothing you can't program around easily.

When the compiler encounters the first user error, it turns off any further semantic checking. So e.g. 'n' undefined TAGS require 'n' compilations to get them out.

A big improvement would be a 'real' loader, support of split I&D space, and separate compilations ('preludes' or equivalent). A new loader has been promised but is not yet available.

For small programs, e.g. a first grade course, the compiler is usable.

For more information, contact:
Algol68 distribution manager; Dept. of Computer Science,
Univ. of Manitoba, Winnipeg, Manitoba, Canada; R3T 2N2.

Resumé of a talk by Dr. Steve Bourne, Bell Labs (formerly at Univ. of Cambridge, U.K.), June 13, 1978; Free Univ. Amsterdam.

Topic was the new shell that he designed and implemented for UNIX V7.

Former shell command files that had e.g. an ed(I) command in them took the standard input to that 'ed' also from the same shell file. The new shell will let standard input stay what it was before, i.e. the terminal by default. The old situation is still available, optionally, and is called: 'in line documenting'. The latter has the syntax:

```
ed << !
std input lines to ed
!
```

With this new feature, shell files can be used as filters, have error recovery possibilities (since the ending ! can be found even when ed collapses), they can be parsed without execution, can have multiple input sources, and can have more extensive parameters and command substitutions.

New control flow statements are added to the shell, with conditional clauses driven by boolean values returned by the commands executed under the regime of the shell file.

- `if commands then commands else commands fi`
- ```
  case argument in
          p1) commands ;;
          p2) commands ;;
          ....
  esac
  ```
- `command && command`
- `command || command`
- `while commands do commands done`
- `for name in namelist do commands done`
  e.g.: `for i in 1 2 3 4 do ..... done`
- `for name do commands done`

New parameter notations are added to support all this:

```
$n stands for 'the number of parameters'
$* stands for 'anything', even empty
\X is a quoting mechanism for a single character
  '...' quoting for multiple characters
  "...$parm..." quoting but do substitute parameters
  $r stands for the last return code
  [] test command, e.g. [ $1 <= $2 ] returns a logical value
```

Examples:
1) a shell file 'append' with synopsis: append [from] to
   would look like:

```
case $n in
        1) cat >> $1;;
        2) cat < $1 >> $2;;
        *) echo "Usage...."
esac
```

2) In the V6 shell, one often encounters loops like:

```
: loop
for i in $1 ...
do
    ... $i ...
done
shift
if x$1 = x exit
goto loop
```

which may now be written as:

```
for i do
    ... $i ...
done
```

Except for the return code, there are even more mechanisms to get information back from commands into the shell level:
when d and t are shell constants, and time and date are commands, d = `date` and t = `time` will assign to d and t the strings that the commands time and date produced on their standard output.

After his talk, Dr. Bourne told that UNIX V7 will definitely require a PDP-11 with split I and D space. He was not very happy with that fact himself, and held a small enquete to sollicit the feelings of his audience. He also told that Bell Labs has a running version of VAX-UNIX, using VAX native mode (as contrary to the commercial VAX-UNIX version that is marketed by Interactive Systems Inc.) The version is for internal Bell use, and it might be released to the world, although he did not know when, how etc. With regard to the release of the UNIX Algol68C compiler, designed and being implemented by himself a.o., the same statement was made.

(Jan Hendrik Thomassen, Nijmegen)

* Bell system runs on bare machine
Interactive's version runs under VMS
```

- or -

Reflections of a Scribe

UNIX Conference
Medical Sciences Building
University of Toronto
June 20-23, 1979

This report is a summary of the exciting things people had to say at the Toronto conference. As I was the only person taking notes for this purpose (although I was helped out by copies of speakers' transparencies), the notes inevitably reflect my own personal biases and knowledge (or lack of such). I have tried to convey enough information for those who did not attend the conference to get a good idea of the principal features and availability of the systems and projects the speakers discussed. At the same time, I hope those who attended can use this report to remind them of all the thrilling and not-so-thrilling things they heard.

I do not guarantee that what I report was actually said. If you want to be SURE, check with the speaker in question. My apologies to everyone whom I have misquoted.

My thanks to Sandra Wright for taking the notes of the last session (Saturday morning). Thanks also to Jim DelGrande, Dave Galloway, Gerardo Lastra, David Miller, Sanand Patel, Rob Pike, Bill Reeves, Henry Spencer, Dr. Martin Taylor and my wife Simone for proofreading and helpful comments on the draft.

June 29, 1979

David Sherman
Computer Systems Research Group
University of Toronto
Toronto, Canada M5S 1A1

1	Pascal and EM-1	Andy Tanenbaum
2	Pascal and EM-1	Garry Fostel
3	Euclid	James R. Cordy
4	Path Pascal	Richard Bolocca
5	Languages for the VAX-11/780	Bill Joy
6	C Compiler for the Z80	Keith Davis
7	UNIX Version 7	Brian Kernighan
8	VAX UNIX 32V	Tom London
9	VAX UNIX - A User Comments	Ed Desautels
10	KSOS – Secure UNIX	Howard S. Weiss
11	What's cooking on the Berkeley VAX	Bill Joy (again)
12	News from Western Electric	Al Arms
13	USENIX	Lou Katz, Melvin Ferentz
14	Winter Conference	John Donnelly
15	UNIX without a UNIX license	Mark Krieger
16	What's on the Berkeley tape	Bill Joy (one more time!)
17	YASL	David Lilienfeld
18	Core Graphics in C	Dennis Munaugh
19	Perception and Information Enhancement	Martin Tuori
20	GPAC and veat	Ron Baecker
21	NYIT Graphics	Tom Duff
22	Graphics for lots of different terminals	Mike Muuss
23	Versatec Typesetter Emulator	G.E. Toth
24	Theories about Office Automation	David Macfarlane
25	Musical Interlude	Bill Buxton
26	Software Register proposal	Lynn Brock
27	Database for a Micro	John Kornatowski
28	Mini-UNIX on the LSI-11	Bob Hudyma
29	Real-time Data Collection and Failure Analysis	Neil Groundwater
30	How to get more out of your 11/70	Dan Gielan
31	Real-time data gathering on UNIX	Eric Ostrum
32	Networking at Purdue	Bill Croft
33	Networking at NYIT	Bill Lindemann
34	RT/EMT: an RT-11 emulator on UNIX	Mike Tilson
35	Software Tools Users Group Report	Dennis Hall
36	Anyone doing system performance monitoring?	Phil Poulos
37	USENIX Committees	Lou Katz
38	11/40 Kernel Multiple Address Space	Alfred Whaley
39	UNIX for 1100 Users??!!!	Ian Johnstone
40	Another large UNIX system	George Goble
41	Office Use of UNIX	Walter Lazear
42	Implementing C and UNIX more efficiently	Carl D. Howe
43	UNIX on a UNIVAC V77-800	Harold Pierson
44	An Accounting System for UNIX	Robert N. Jesse
45	A High Performance UNIX System	Mike Muuss
46	UNIX on the IBM 370	Steve Bellovin
47	UNIX in the Undergraduate Lab Environment	Don Schertz
48	UNIX in a large educational environment	Mike O'Dell
49	QED, or The Little Ed That Grew	Robert Pike, David Tilbrook
50	News from the U.K.	R.P.A. Collinson
51.	What's happening at BBN	Carl D. Howe
52	Tuning PWB UNIX	George Pajari
53	Screen editors	Mark Pearson

Name	Topic	Speaker
Arms, Al	News from Western Electric	12
Baecker, Ron	GPAC and veat	20
Bellovin, Steve	UNIX on the IBM 370	46
Bolocca, Richard	Path Pascal	4
Brock, Lynn	Software Register proposal	26
Buxton, Bill	Musical Interlude	25
Collinson, R.P.A.	News from the U.K.	50
Cordy, James R.	Euclid	3
Croft, Bill	Networking at Purdue	32
Davis, Keith	C Compiler for the Z80	6
Desautels, Ed	VAX UNIX - A User Comments	9
Donnelly, John	Winter Conference	14
Duff, Tom	NYIT Graphics	21
Ferentz, Mel	USENIX	13
Fostel, Garry	Pascal and EM-1	2
Gielan, Dan	How to get more out of your 11/70	30
Goble, George	Another large UNIX system	40
Groundwater, Neil	Real-time Data Collection and Failure Analysis	29
Hall, Dennis	Software Tools Users Group Report	35
Howe, Carl D.	Implementing C and UNIX more efficiently	42
Howe, Carl D.	What's happening at BBN	51
Hudyma, Bob	Mini-UNIX on the LSI-11	28
Jesse, Robert N.	An Accounting System for UNIX	44
Johnstone, Ian	UNIX for 1100 Users??!!!	39
Joy, Bill	Languages for the VAX-11/780	5
Joy, Bill	What's cooking on the Berkeley VAX	11
Joy, Bill	What's on the Berkeley tape	16
Katz, Lou	USENIX	13
Katz, Lou	USENIX Committees	37
Kernighan, Brian	UNIX Version 7	7
Kornatowski, John	Database for a Micro	27
Krieger, Mark	UNIX without a UNIX license	15
Lazear, Walter	Office Use of UNIX	41
Lilienfeld, David	YASL	17
Lindemann, Bill	Networking at NYIT	33
London, Tom	VAX UNIX 32V	8
Macfarlane, David	Theories about Office Automation	24
Mumaugh, Dennis	Core Graphics in C	18
Muuss, Mike	A High Performance UNIX System	45
Muuss, Mike	Graphics for lots of different terminals	22
O'Dell, Mike	UNIX in a large educational environment	48
Ostrum, Eric	Real-time data gathering on UNIX	31
Pajari, George	Tuning PWB UNIX	52
Pearson, Mark	Screen editors	53
Pierson, Harold	UNIX on a UNIVAC V77-600	43
Pike, Robert	QED, or The Little Ed That Grew	49
Poulos, Phil	Anyone doing system performance monitoring?	38
Schertz, Don	UNIX in the Undergraduate Lab Environment	47
Tanenbaum, Andy	Pascal and EM-1	1
Tilbrook, David	QED, or The Little Ed That Grew	49
Tilson, Mike	RT/EMT: an RT-11 emulator on UNIX	34
Toth, G.E.	Versatec Typesetter Emulator	23
Tuori, Martin	Perception and Information Enhancement	19
Weiss, Howard S.	KSOS – Secure UNIX	10
Whaley, Alfred	11/40 Kernel Multiple Address Space	36

WEDNESDAY MORNING

Session 1: LANGUAGES ON UNIX

Chair: Ron Baecker, University of Toronto

Speaker 1, at 9:15 a.m. **Pascal and EM-1**

Prof. Andrew S. Tanenbaum
Vrije Universiteit – Wiskundie Seminarium
Box 7161
1007 MC Amsterdam, The Netherlands

Andy Tanenbaum gave a detailed talk on the UNIX Pascal system developed at the Vrije Universiteit by Johan Stevenson, Hans van Staveren and himself. Their aim has been to make the compiler and support software as portable as possible, with transport to various mini's and micro's in mind. They feel that "C is not the beginning and end of the world," and that Pascal's attractiveness lies in its simplicity and readability.

The Pascal language implemented is the complete language, and is compatible with the "British Standard," which has been proposed as an international (ISO) standard. There are also a few extensions, such as assertions, mark/release, external procedures, and UNIX-style zero-terminated strings. It is also possible to create libraries of separately compiled Pascal, C, or assembly language procedures that can be called from Pascal programs.

There are also many debugging and performance facilities, such as warning messages for unused or undefined variables; run time error messages giving source line number; counts of source line executions; and procedure entry/exit tracing.

The system can be used to produce either compiled or interpreted code: "pc prog.p" will produce interpreted code, and "pc -C prog.p" compiled code. In both cases the compiler produces code for an abstract stack machine called EM-1, described in *CACM*, March 1978. If the -C flag is given, the EM-1 code is passed through a machine-independent optimizer, and then translated to the UNIX assembly language to be assembled and loaded by as and ld. If the -C flag is absent, the EM-1 code is assembled to EM-1 binary for subsequent interpretation. There are 64 versions of the interpreter corresponding to 6 independent debugging options. The system automatically selects the correct version from a directory, if it is found, or creates and saves a new one dynamically, if need be.

The optimizer's capabilities include constant folding (e.g. mapping 10+4 into 14), using special instructions (e.g. increment for i:=i+1), strength reduction (e.g. shifting for multiplication), reordering (e.g. -k/8 becomes k/-8), and many more.

The distribution is available as part of the 1979 conference distribution. It contains all the sources to the compiler proper, the optimizer, the EM-1 to PDP-11 translator, and the interpreter, as well as various support programs. There are also libraries, including one containing all the UNIX system calls, so it is possible to call open, seek, fork, exec, etc. from Pascal programs. The tape also contains versions of *as* and *ld* capable of handling files larger than 64K bytes. Separate I and D space is useful, but not required.

Anyone who installs the system should write to Andy to be put on the mailing list for bug reports (the first of which was available at the conference). As soon as the version 7 Pascal system becomes available (in the fall) both it and the version 6 one will be distributed. To avoid sending tapes across the ocean, the Vrije Universiteit has made an

not have the conference distribution.

Speaker 2, at 10:00 — Pascal and EM-1

Garry Fostel Intermetrics Incorporated 701 Concord Avenue Cambridge, Mass. 02138

Garry Fostel discussed the Pascal compiler described above. He said it is exceptionally well put together and documented. It is available in the United States from Intermetrics for a $50.00 handling charge.

It runs on standard UNIX V6, and there are certain assumptions about file names near the root. There may be a version for PWB soon.

coffee

Speaker 3, at 10:45 a.m. — Euclid

James R. Cordy
Computer Systems Research Group
University of Toronto
Toronto, Canada M5S 1A1

The Toronto Euclid Compiler Project is a joint project of I.P. Sharp Associates (better known for their API. services) and CSRG, funded by the U.S. Defense Advanced Research Projects Agency (DARPA) and the Canadian Department of National Defence (DND). It is a follow-up on an attempted implementation by the Systems Development Corporation, which failed in an earlier attempt to produce a Euclid compiler.

Euclid is the so-called "language of secure systems", where secure means *provably* secure. The Euclid language is roughly based on Pascal but has many extensions. It is *very* strongly typed; it has explicit visibility and scope control, unlike Pascal; it prohibits aliasing, or having two identifiers for a single object. This is necessary for program verification and extends even to assuring that no two actual parameters to a routine address the same element of an array.

The heart of the language is the *module*. A module operates as a single mechanism which manages a set of data structures internally and "exports" only a clearly defined set of externally visible operations and data. The program external to the module can invoke any of the exported operations of the module and can access the exported data, but knows nothing of the internal implementation of the module. Thus, the module looks like a "black box" which provides certain services and information through its exported operations and data. It is a self-contained "package" of procedures, variables and constants.

Types can be parameterized in Euclid. A parameterized type definition defines a template for a "family" of types defined by different parameter values. In its simplest form, a parameterized type can define a set of range types whose upper and/or lower bound depends on a parameter value. The full power of parameterized types is realized in the *parameterized module*, which can define a whole family of similar mechanisms.

expression, but is "constant" in the sense that, once set, the value remains constant in the scope.

Euclid provides pre- and post-assertions for routines, which enable the specifications and assumptions of a routine to be encoded and checked at run-time. Modules may have "invariant" assertions which specify invariant conditions of the module. These also can be checked at run-time when entering and exiting routines exported from the module.

Pointers in Euclid are restricted to a "zone" of storage maintained either by the system or by a user module, and to a particular type of object (which may be paramaterized).

The project schedule so far has been:

Sept. 1977 – Transliterator of Small Euclid to C – so as to bootstrap by writing the entire Euclid compiler in Euclid.
June 1979 – Translator – compiler for a large subset of Euclid.
Fall 1979 – Compiler should be finished. It will cover most of Euclid, and will include all verification features. Implementation of an automatic verifier is not within the scope of the compiler project.

The compiler does not run without separate I and D space. There are 4 machine-independent and 2 machine-dependent passes to the compiler. Performance: Object code produced is generally as good as or better than C in space and time. Because there is so much modularization in Euclid, procedure calling has been carefully optimized. A call to a parameterless procedure with no local variables costs only two instructions (JSR, RTS). The source code for the Euclid Translator is about 60,000 lines. To compile the compiler takes about six hours (two to three hours for the largest pass; there are six passes) on a relatively quiet 11/50. No attempt has been made to tune the compiler for speed.

Euclid is not available publicly yet, although the current version is being delivered to DND (Canada) and DARPA. When the project is complete, it is hoped that IPSA will distribute it commercially (with full maintenance) and that it will be available (without support) to educational and research institutions from CSRG. If you would like to be placed on the mailing list for information about distribution of the Euclid Compiler when it becomes available, send your name to Jim Cordy (address above).

Speaker 4, at 11:35 — Path Pascal

Richard Bolocca
University of Illinois
Champaign, Illinois

Path Pascal is a superset of Pascal P. It has additional features for concurrency; for data encapsulation; and for self compilation. It is used for the teaching of systems programming at the University of Illinois. It runs on a Cyber and a PDP-11, and may be used for real-time programming by the NASA space shuttle program. Path Pascal was developed by Roy Campbell at the University of Illinois.

Concurrency: features are the *process*, which is like a procedure but "goes its own way"; and interrupt processes. Data encapsulation: there is a new type, called an "ob-

ject", much like a module in Euclid or Modula; processes can be created dynamically with this.

The compiler is self-compiling and takes two passes to reach assembler code. The compiler source is about 4K lines for each pass; the binary takes 48K and 32K bytes for the two passes. The second pass compiles itself in about 4-5 minutes. The compiler needs separated I and D space to run, or non-separated I and D with "segmentation" overlays.

The language currently runs on UNIX but can also run standalone on an LSI-11. It is highly portable: in existence now are compilers for the Z80, all PDP-11's, and the Series/1. Being developed is one for the Prime 500 and similar models.

Path Pascal will be available by the fall for a nominal (tape-handling) fee.

Speaker 5, at 11:45 **Languages for the VAX-11/780**

Bill Joy
University of California at Berkeley
984 Riley Drive
Albany, California 94706

Bill discussed the various languages and near-languages running under UNIX on the VAX at Berkeley:

1) *Pascal.* The Berkeley Pascal is well known. Like the front end of the portable C compiler, its front end is relatively machine independent. The program *pi* (Pascal interpreter) is being modified to become *pc*, which will output code to be compiled by either *pcc* on an 11 (generating PDP-11 assembler files) or *pcc* on the VAX (generating VAX assembler).

2) *LISP.* The LISP compiler, a.k.a. "Franz Lisp", is written in C. A simple interpreter is running, but the system so far can run only 2/5 of MACSYMA (two of five parts of the MACSYMA compiler which take up 400K and 350K bytes respectively have been run together on a ½-megabyte machine). Work is now proceeding on a "Byte LISP" for compactness, and a compiler.

3) *APL.* Work is being done on moving Ken Thompson's APL to the VAX. The advantage of running it on the VAX is that one can get around the 65K workspace limitation (with paging) and run real APL.

4) *Modula.* The compiler, originally BCPL code from York (England), was rewritten line for line to compile as C, and has been taken over to the VAX. It does not yet compile VAX code.

5) *Rigel.* This relational database language was described in the proceedings of the last SIGMOD conference (at Boston).

6) *MACSYMA.* This system is supported by the LISP system described above.

The following are languages which the Berkeley people are thinking about and/or interested in:

7) *3.* A modified version of FORTH. Since FORTH is proprietary, no more need be said...

8) *Algol 68.* They would like to move this to the VAX.

9) *Snobol/Spitbol.* It would be especially nice to get a Spitbol up.

10) *STAPLE.* This is being worked on. It is a structured applications and programming language for modification of system programs. It's based on C, PL/1, Pascal....

None of the above languages is available officially (i.e., nothing is packaged) except for the Pascal interpreter.

The following can only marginally be considered languages...:

11) *Termcap.* Makes the screen editor terminal-independent.

12) *C Shell.* A shell with its own command language, much like C.

Speaker 6, at 12:15 **C Compiler for the Z80**

Keith Davis
Teletype Corporation
4 Mayflower
Vernon Hills, Illinois 60061
(213) 982-3619

The C compiler for the Zilog Corp. Z80 is a cross-compiler which runs under PWB. Developed by Interactive Systems, it features everything in the C compiler except floating point. It generates specific Z80 code, which is 42% larger than PDP-11 code because of instruction inefficiency.

The compiler is available, along with a debugging package (like cdb), for those within the Bell system from Keith Davis of Teletype. Internal charge: $5,000. Outside the Bell system, it's available from Interactive Systems (Keith can point you at the right people).

LUNCH

WEDNESDAY AFTERNOON
Session 2: THE UNIX OPERATING SYSTEM
Chair: Bill Reeves, University of Toronto

Speaker 7, at 2:00 p.m. **UNIX Version 7**

Brian Kernighan
Bell Labs
Murray Hill, New Jersey 07974

A lot of the Version 7 changes were covered in the BSTJ issue on UNIX. The important new things are: the shell; huge files (2^{30} bytes); portability; the portable C compiler; lint (C program checker); stdio package; Fortran 77 compiler; make; lex (lexical analysis phase of yacc); awk (pattern scanning and processing language); sed (stream editor); learn (computer-aided instruction about UNIX); adb (a "complex but indispensable" C debugger); uucp (UNIX-to-UNIX communications handling).

The new shell is much more oriented to programming than previously. Gone is the old "goto"; in its place are "for ... do ... done ", "case ... esac", "if ... then ... else ... fi", "while", "until", and a "trap" command to handle interrupts. Also new are shell variables (including special ones for home directory, mail file and bin path search).

Changes in C for portability are all described in the C book; in addition, structure assignment has now been implemented, and there is an "enumerated type" feature.

Lint is a program checker that goes right through a program (including multi-file programs) and checks for type violations, portability problems, probable errors, and bad style that may be evidence of error.

Make is a command to compile according to instructions in a "makefile". It knows about a lot of things by default, such as that "cc -c zork.c" will produce "zork.o". Its only drawback it that it does not yet handle ar libraries.

F77 compiles the complete Fortran 77 language, with a few extensions. It generates the same intermediate code as C, so UNIX I/O is accessible. Also on the distribution is struct, a program to convert standard unintelligible Fortran to Ratfor.

Awk is a pattern scanning and processing language in which programs are typically 1 or 2 lines. Initialization and declarations are not used -- awk decides what an item is by how it is accessed and used. It is useful for such tasks as switching two fields within a file, or adding up the contents of a particular field.

Learn interprets a script to teach UNIX. Scripts are available for teaching UNIX file handling, ed, C (not a very good tutorial, according to Brian, who wrote it), and the –ms macro package (the Bell people don't think anyone should use nroff directly so there's none for nroff). The lessons are streamed and tune themselves to the competence of the student. Brian mentioned that writing scripts for these things is extremely hard.

Tar, the new tape program, was discussed by Tom London (below, but the information belongs here). Files sent to tape with tp still work. All code is written on the tape in ASCII, so any system can read it. You can block the tape with huge blocks. Updates on the end of the tape are allowed, and extracts are done on the last instance of a file. Directories are built for you when you extract files.

coffee

Speaker 8, at 3:30 **VAX UNIX 32V**

Tom London
Bell Labs
Murray Hill, New Jersey 07974

UNIX 32V is UNIX on a VAX-11/780. Tom described in some detail the mysteries of VAX addressing, with its 31½ bit address space. Presently running on the VAX is a full UNIX V7, with the same file system, shell, commands, language and system interface as the PDP-11. There are just a few incompatabilities: the VAX word size is 32 bits, so programs which know that a word is two bytes are wrong and ints are aligned at 4-byte instead of 2-byte boundaries; long ints are encoded in the reverse order from the PDP-11; traps are handled differently; pointers cannot conveniently be assigned the value -1; external variables declared contiguously won't necessarily be contiguous in memory; and so on.

The VAX, with its block-move instruction, performs faster than the PDP-11 in tasks which involve moving large chunks of data at once (although it does not quite live up to the manufacturer's billing of being consistently twice as fast). Compiling a C program on the VAX is faster than on the 11/70, even though the cc being used is Steve Johnson's, which is not as finely tuned as Dennis Ritchie's.

UNIX 32V has partial swapping -- pages can be scattered anywhere in memory, and only part of a program will be swapped out to make space for another. These improvements reduced execution times by a factor of 2/3. Demand paging is not yet available, but is being worked on at Berkeley (see Bill Joy's talk below).

In summary, the VAX is well worth using as a UNIX machine. You get a full V7 on a faster machine, a larger address space, and the capability of handling huge programs. UNIX 32V is now available from Western Electric (see Speaker #12 for details).

Speaker 9, at 4:30 **VAX UNIX - A User Comments**

Prof. Ed Desautels
Computer Sciences Department
Wisconsin University
1210 W. Dayton
Madison, Wisconsin 53706

The VAX at Wisconsin is now running UNIX V7. It is part of a network of several PDP-11's and LSI-11's, hooked up to each other in various ways. The VAX originally ran VMS (with auto-reboot); the VMS Pascal being developed at University of Washington (Seattle) was tested at Wisconsin.

VAX UNIX was found to be easy to install, although it is "not yet robust". Projects underway on VAX UNIX include: an intelligent mass storage system (Dave DeWitt, Tony Klug); compilers and operating systems for computer networks (Raphael Finkel, Marvin Solomon); a data base system for AI (Larry Travis); a Computer Science network for Telenet (Larry Landweber, Ed Desautels); connection with a large array of micros for perception research (Len Uhr); new architecture and systems work (Ray Moore, Bob

Cook, Ray Bryant and Goodman); and facilitation of access to computing for the visually impaired, along with work on the TeleText/Viewdata systems (Ed Desautels).

Speaker 10, at 4:45 KSOS -- Secure UNIX

Howard S. Weiss
U.S. Department of Defense
9800 Savage Road, Room 171
Fort Meade, Maryland 20755

KSOS stands for Kernelized Secure Operating System. It started as an ARPA project in 1977 and is being coded in Modula. (Euclid was the original choice but the Euclid project was not ready in time.) Written by Ford Aerospace at Palo Alto, it should be completed this winter.

KSOS is an operating system in itself. The "UNIX" that runs on it is a UNIX emulator. Other emulators may be written, should the demand warrant it. The UNIX emulator is committed to performing at no worse than one-half the speed of UNIX V6.

KSOS will be available to those within the U.S. federal government system. It is unknown yet whether it will be made publicly available.

Speaker 11, at 5:00 What's cooking on the Berkeley VAX

Bill Joy (again)
University of California at Berkeley

Projects underway on the VAX called "Ernie" (½ megabyte) include:

1) paging (Ozalp Babaoglu, Juan Porcar) -- see below
2) microcode work (Dave Paterson, Richard Tuck)
3) a new floating point box (George Taylor, V.V. Kahan)
4) a floppy-disk driver (Richard Tuck)
5) swapping to UNIBUS disks (Eric Allman, Bob Kridle)
6) LNI interface for networking (Ken Birman, Larry Rowe)

Because the hour was late, Bill restricted himself to talking about paging. The goal of the pager, called PUNIVAX, is to support large programs, with mostly large text segments (up to 6 megabytes of text). There were some clear architectural problems to overcome, in that the VAX has no reference bits and has very small pages (512 bytes). The paging system is well on its way now, however.

THURSDAY MORNING
Session 3: Items of Interest to Users
Chair: Mel Ferentz, The Rockefeller University

Speaker 12, at 9:00 a.m. News from Western Electric

Al Arms
Western Electric Company, Inc.
Box 25000
Greensboro, North Carolina 27420

Al reminded us all that Western Electric sells a UNIX license "as is", with no warranty of any sort and no maintenance or service. He announced that UNIX Version 7 and VAX UNIX 32V are on the market. Western Electric's commercial rates are now:

System	1st CPU	Add. CPU	Binary
Mini UNIX	$12,000	$4,000	
UNIX V8	20,000	6,700	8,400
PWB 1.0	30,000	10,000	12,000
V7	28,000	9,400	11,700
32V	40,000	15,000	16,000

For users who already have a UNIX license, V7 may be obtained for $12,000, and $4,000 for additional CPU's. No such discount is available for 32V. Educational institutions may get an "administrative" license, for internal business uses, at one-third of the commercial rates. The educational research price is $300 for V7 or for VAX 32V; $230 for V8 tape and manuals.

PWB Version 2.0, which has the V7 file system and is fully compatible with V7, was just released internally within Bell. It will not be available publicly for some time.

MERT will not be released (i.e., it officially does not exist any more). V7 will probably not be released for the Interdata 32-bit machine (8/32). A point of interest: there are currently about 800 V8 licenses, of which 250 are commercial.

Speaker 13, at 9:45 USENIX

Lou Katz
Columbia University
New York, NY

Melvin Ferentz
The Rockefeller University
1230 York Avenue
New York, NY 10028

At the users' meeting in New York last year, a committee of five was nominated to investigate the setting up of a formal UNIX Users' organization to handle such matters as tape distribution, newsletter publication and conference planning.

On June 20, 1979, an association was formed called USENIX. It has a legal existence in the State of New York. Its Board of Directors initially consists of: Lou Katz, President; Lew Law, Vice-President; Armand Gazes, Secretary; Mel Ferentz, Treasurer; Mars Gralia and Peter Weiner, members of the Board. USENIX invites institutions and individuals to join; it has no exclusive "rights" over anything and no-one is obliged to join.

If USENIX becomes successfully operational, it will set up in business at the Rockefeller University, handling the affairs described above.

Membership in USENIX will fall into one of four classes: (a) voting; (b) individual; (c) public individual; (d) non-voting institute. Voting members will be institutes with a UNIX license who pay $300 for each vote, to a maximum of the number of CPU's for which they hold licenses. The educational-institution rate will be $100 instead of $300. Voting members will delegate individuals to exercise their institution's vote. Individuals will be permitted to join for $12 and receive the newsletter, which will be produced at least 10 times per year. Individuals who work for or are associated with a UNIX license holder and are therefore bound by the license's non-disclosure clauses will hold regular individual membership and be entitled to receive information about proprietary parts of the system. Those not bound by non-disclosure may hold an "outside" membership. Non-voting institutional membership will be available to those in the Bell system (who are of course not "licensed").

USENIX will hire a full-time employee to handle the newsletter, tape distribution and the maintenance of a data base on the UNIX. The facilities at the Rockefeller University will be available for this purpose.

The bylaws and an invitation to join will be distributed to all parties thought to be interested. Election of a new slate of directors will be held by mail, probably around the December. The association may be contacted by mail at:

USENIX Association
Box 8
The Rockefeller University
1230 York Avenue
New York, NY 10021

Lou announced that the Association is currently asking for volunteers to serve on four committees: Agenda for the Winter Conference at Boulder; Site Selection for next year's Summer conference; distribution tape format; and nominations for office for December elections (but see Speaker #37, below).

Speaker 14, at 10:30 — **Winter Conference**

John Donnelly
National Center for Atmospheric Research
Boulder, Colorado

A brief announcement that the Winter Conference will be held January 29 - February 2 (Tuesday through Saturday) at a convention centre in Boulder. All users are encouraged to bring their /dev/ski drivers.

coffee

Speaker 15, at 11:00 — **UNIX without a UNIX license**

Mark Krieger
Whitesmiths Limited
127 East 59th Street
New York, NY 10011
(212) 799-1200

Whitesmiths have started from a UNIX binary license and created their own C compiler for the PDP-11. The compiler produces either UNIX or Macro-11 assembler. It supports the full Version 7 compiler as defined in the Kernighan and Ritchie book. (Extensions to that syntax mentioned by Brian Kernighan in his talk, such as structure assignment, will be provided for if these become standard.) The compiler runs under any of UNIX, RSX-11M, RT-11, IAS and RSTS.

A portable library comes with the C compiler. It features alloc and free; char, line and formatted I/O; and string functions. Stdio is not followed too closely.

A PDP-11 machine library provides certain functions that the hardware can't do on some machines, such as floating-point arithmetic.

Currently available is a C compiler for the 8080. It runs on any PDP-11 and takes the same V7 C as input. It generates an intermediate assembly language called "a-natural". The translator from a-natural to ISIS-II or CP/M microsoft assembly code is the fourth pass of the compiler. It generates code only 50% larger than the corresponding PDP-11 code (not bad considering the instruction inefficiencies in the 8080). The machine library for the 8080, written by Bill Plauger, supports all 16-, 32- and 64-bit operations supported by C.

By July 1979 Whitesmiths will be distributing a full "a-natural" assembler, loader and librarian for the 8080. It will run on the PDP-11 or the 8080 on any of the operating systems named above.

By the fall the IDRIS operating system (named after the Persian deity over toolsmiths) will be on the market. It will look exactly like UNIX and will be runnable on an LSI-11. Memory management should be available by the early winter.

By early 1980, a native C compiler for the VAX and C compiler for a 16-bit micro should be available.

Mark went into some detail about the features of the "a-natural" assembly language. The C compiler currently available costs $5,000 (including source). Without the source, it's $500.

Speaker 16, at 11:55 — **What's on the Berkeley tape**

Bill Joy (one more time!)
University of California at Berkeley

The tape includes: Berkeley Pascal version 1.2; *ex* (display editor which requires separate I and D space) version 2; C shell (runs on either V6 or V7, good while converting); –me macro package (faster than –ms); a new mail program; the Berkeley networking (for machines hooked up by back-to-back DZ-11's; mods to stdio for simultaneous read/write; miscellaneous utilities; and a V7 simulator library (system dependen-

cies can all be left in this lib and changed at once).

For each item one will find: full source; V6 binaries; V7 makefiles; all manual sections and documents; and a Versatec copy of the printed documentation: Everything except Pascal runs on both VAX UNIX and V7. Everything runs on 11/40's and 11/34's.

To get the tape you need $60, a copy of your UNIX license, and a V7 or Phototypesetter license for the –me package. You will get a 1200-foot tape at 800 bpi.

Speaker 17, at 12:10 — YASL

David Lilienfeld
Johns Hopkins University
34th & Charles Streets
Baltimore, Maryland 21218

YASL – Yet Another Statistical Language -- was designed because there was a need for a statistical package on a small machine. The reasons existing packages (such as BMD or SPSS) could not be used were (a) they are clumsy; (b) you "can't tell them what you want them to do"; and (c) you need a large machine, such as a DEC-10 or a CDC 6600, to run them because of the memory they take up.

The writers of YASL have emphasized structure in their language, which they have tried to make a superset of C. Certain features useful for matrix statistics have been designed, such as case ranges (e.g., case 'a'-'z':), and declarations for various types of matrices. Matrices can be dynamic and/or virtual. Virtual matrices are necessary to run large regression analyses and the like on a machine with rattively little memory.

The compiler is currently being written. It is the aim of the authors to have something running by the end of the summer, and a finalized version by June 1980. At that point it will become generally available (including the source).

LUNCH

THURSDAY AFTERNOON
Session 4: Graphics, Music and Typesetting
Chair: Ron Baecker

Speaker 18, at 1:55 p.m. — Core Graphics in C

Dennis Mumaugh
U.S. Department of Defense
9800 Savage Road
Fort Meade, Maryland 29755

Core Graphics is an "Independent Graphics System" designed by the writers of the Core Report (see *ACM Computing Surveys*, December 1978 for a whole issue on Core). It features a set of common subroutines for any language to do all the basic things a graphics language should do. At the bottom level are device-dependent primitives to do such tasks as basic line drawing.

Dennis reported that a Core Graphics for C has been written, although it is not yet complete. It has been checked on the Version 7 C compiler. The only device driver with it is one for a Genesco graphics system (which might not be of help to anyone, since Genesco itself has no standard graphics format). It differs from Core in a few details, such as not having function names longer than 7 characters (the maximum distinguished in C).

This graphics package is available as part of the conference distribution. The Core manual is not on it; that may be obtained from SIGGRAPH on machine-readable tape. This tape has a list of differences between the package and the SIGGRAPH manual. Some test programs and instructions on how to build the driver are supplied.

Speaker 19, at 2:05 — Perception and Information Enhancement

Martin Tuori
Defence and Civil Institute of Environmental Medicine (DCIEM)
1133 Sheppard Ave. West
Downsview, Ontario, Canada M3M 3B9
(416) 633-4240 x204

Martin Tuori and Dr. Martin Taylor are doing work at DCIEM on human perception. Their aims are to study vision and audition in complex situations, and to develop visual and auditory enhancements which will aid persons working in those situations. Experiments will be conducted to demonstrate the effectiveness of such enchancements. The techniques will be of use with information from various sources, including earth resources satellite imagery.

Martin discussed the "IPAC" file format for storage of picture, sound and other data, and invited interested persons to contact him. The format is hierarchical; the structures inherent in the data are first presented in a header. This will permit programs to interpret data files of varying structure. He would like to discuss the IPAC technique, with a view to working out a more widely accepted standard.

Martin then showed slides, generated with a Dicomed D48 film recorder, including: (a)

photographs of parts of Canada from LANDSAT satellite, and various techniques by which they can be enhanced; (b) images made with *fns*, a program written by Mike Tilson at the University of Toronto to generate coloured patterns based on x-y-r-theta functions and a 256-colour map; and (c) pictures of Mars taken by Viking orbiter and brought from Caltech by Rob Pike.

Speaker 20, at 2:20 **GPAC and vcat**

Ron Baecker
Computer Systems Research Group
University of Toronto
Toronto, Canada M5S 1A1
(416) 978-6708

There is nothing new to be said about GPAC. It continues to be used as the graphics package at the University of Toronto for building highly interactive systems (such as ones for producing animated film, composing and performing electronic music and visualizing the behaviour of simulation models). It is available and fully documented as part of the Toronto distribution. GPAC was designed and written by Bill Reeves, who also wrote *vcat*, a Versatec typesetter-simulator which runs as an output filter to troff. Vcat (which was used to generate this document) is available free as part of the University of Toronto distribution #3.

Speaker 21, at 2:30 **NYIT Graphics**

Tom Duff
Computer Graphics Lab
New York Institute of Technology
P.O. Box 170 Old Westbury, NY
(516) 626-0938

The Computer Graphics Lab at NYIT is in the business of making cartoons, ranging from short spurts to (allegedly) full-length features. They don't distribute any of their software. They have lots of hardware: 10 PDP-11's which run UNIX, lots more which don't, a VAX, 18 frame buffers (512 x 512 x 8 bits each), and, as the ultimate peripheral device.... a TV studio. Need we say more?

Tom showed a very entertaining 20-minute videotape about NYIT's graphics.

coffee

Speaker 22, at 4:00 **Graphics for lots of different terminals**

Mike Muuss
Johns Hopkins University
Electrical Engineering Department
Barton Hall
Baltimore, Md. 21218

Mike Muuss described the "Terminal Independent Graphics Package" at Johns Hopkins (funded by the Army Research Office). The goal of the project was to create a package which would let one create a graphical display only once, and use it subsequently on different terminals without re-executing the original "a.out".

The system has a graphics format which lets one add new devices easily. It is also possible to reprocess the display after generation. The display size is exactly the same for a given picture on all devices.

Low level primitives, using the concept of a "virtual pen", have been written for the package: pen up/down, move pen, new form, new origin, set colour/intensity. High-level routines available include rotation, scaling, and line, symbol and number drawing. 3D perspective is under development.

Devices currently supported with the package include the HP2000A series; Tektronix 4006, 4010 and 4014-1; Hewston Instruments "Complot"; Versatec; Diablo 1620 series; and Ramtek Colour Graphics. It takes one afternoon's work to add a new device to the package.

The package is distributed by:

Dr. Bruce Henriksen
Ballistics Modelling Division
Ballistic Research Lab
United States Army, Aberdeen Proving Ground
Aberdeen, Maryland 21005

Speaker 23, at 4:10 **Versatec Typesetter Emulator**

G.E. Toth
Johns Hopkins University
Electrical Engineering Dept.
Barton Hall

VERSET was partially developed at Johns Hopkins University but completed and debugged for the commercial firm which markets it. It can run without separate I and D space and fits entirely on an RK disk. The font file takes up 1321 blocks and covers all troff point sizes and widths. VERSET runs as an output filter to troff without any modifications to existing software. It can process a page on the Versatec in 20 seconds.

Verset is available in binary form for $2000 to commercial users, $500 to educational licensees. For more information contact:

RLG Associates, Inc.
Suite 18
11250 Roger Bacon Drive
Reston, Virginia 22090
(703) 471-1100

Speaker 24, at 4:25 — **Theories about Office Automation**

David Macfarlane
Bell-Northern Systems Research
522 University Avenue
Toronto, Canada M5G 1W7

David Macfarlane gave us a slide show from BNSR called "An Integrated Methodology for Office Automation". In the slide show, the merging of the three fields of computers, communications and office equipment is discussed, and certain design problems in the construction of an "Office Information Communications System" are explored.

The bottom line is that there is a need for an integrated perception for the study, design and implementation of an automated office system. Nothing has been done beyond the thinking stage as yet.

Speaker 25, at 4:50 — **Musical Interlude**

Bill Buxton
Structured Sound Synthesis Project
Computer Systems Research Group
University of Toronto
Toronto, Canada M5S 1A1

Bill Buxton "conducted" a half-hour concert generated by the SSSP music synthesizer running off an LSI-11 with floppy disks. The entire system is written in C and takes up about 14.5 Kwords of core. The musical patterns were loaded into core at the start of the performance, and Bill "conducted", sitting at a terminal with a bit pad, cursor and function box. The terminal's addressable cursor was used to approximate the graphic environment available on CSRG's 11/45 with a full tablet and vector graphic display (those who came to the demos on Wednesday and Thursday evenings saw the real thing). "Conducting" consisted of varying, as a performer, the volume, timbre, articulation, tempo and other properties of the patterns.

World UNIX Reversi Championship

Thursday, June 21, 7 p.m. - 1 a.m., the World UNIX Champion of Reversi was determined. Reversi, as you all now know, is a board game (also known as Othello) played by two players on a 64-square board.

Eight entries competed, including one written by the organizer, Bill Reeves. Our hardware mastermind (Guy Fedorkow) set us up with nine terminals in the CSRG conference room. Dave Sherman directed the tournament as a 4-round Swiss system (along the lines of a chess tournament), in which players with equal records were paired in each round. We ran three games simultaneously on the 11/50 (i.e., six programs, of which a maximum of three were active at any one time), and one on the 11/45 in competition with the graphics/music demos going on in the lab.

R. Dennis Rockwell of Duke University emerged as the winner with a perfect 4 out of 4, beating Tom Duff in the last round. Bill Reeves finished second, losing only to rdr in the third round.

To ensure that the tournament didn't last all night, a time limit of 15 minutes per program per game was enforced. All games were run with time(1), and any program using more than 900 seconds of user+sys defaulted the game. Fortunately, no-one ran over, although rdr did finish his second game with only 8.5 seconds to spare!

The final scores: rdr (Dennis Rockwell, Duke University) 4 points; bill (Bill Reeves, University of Toronto) 3; td (Tom Duff, NYIT), allan (Allan Wilkes, Princeton), sask (Larry Custead, University of Saskatchewan), and ber (Brian Redman, Bell Labs) all 2 points; uwo (team of three from the University of Western Ontario) 1; psl (Peter Langston, New Permanent Wrinkle) 0. Thanks to all competitors and spectators for a great tournament. See you in Boulder?

[Rumour has it that Bill Reeves is planning on holding the first meeting of SIGREV/USENIX in Fiji in the summer of 1980.]

FRIDAY MORNING
Session 5: Interesting projects on interesting UNICES
Chair: Bob Hudyma, University of Toronto

Speaker 20, at 9:15 — **Software Register proposal**

Lynn Brock
Computer Corporation of America
1600 Wilson Blvd., Suite 903
Arlington, Virginia 22209
Washington, D.C.

The Computer Corporation of America does UNIX work as a contractor for ARPA in cybernetics technology and the like. CCA has proposed to ARPA, and expects funding by October for, a "UNIX Software Register", to be known as USR or perhaps /usr (groan), an on-line database of information about UNIX-related software. Both commercial and non-commercial materials would be involved, and the list would include programs written in C and compilers for C, regardless of the machine.

The proposed registry would be accessible via an INWATS (800 exchange) free phone number, as well as on the ARPAnet and perhaps TELENET. Users would be able to dial in and access information, enter new information, and update information entered by them. The entire database would be maintained by the users, with the administrators merely keeping an eye out for irregularities. USR would not distribute any software; it would be up to those wishing to get it to contact the individual suppliers.

The suggestion was fairly well received by those present. In the light of the creation of the USENIX association, it would be appropriate for these two groups to get together to work out how best to handle the database.

Speaker 27, at 9:30 **Database for a Micro**

John Kornatowski
Computer Systems,Research Group
University of Toronto
Toronto, Canada M5S 1A1

MRS is the Micro Relational System for information retrieval on UNIX and UNIX-like systems. It is designed for fairly small data bases (up to 10 megabytes). It features: easily-defined data bases, a powerful retrieval facility, simple data entry and interactive data modification. MRS is fully interfaced to UNIX and has been tested in actual applications.

MRS runs on UNIX, Mini-UNIX and LSX, and has been tested by the designers on LSI-11's and PDP-11's of every size. The data can be stored on anything from a floppy to a big disk. The total size of all programs in the package is about 300 blocks, so users can allocate whatever space they have left for data.

John gave us a sample session with MRS in his slides, to be followed by hands-on demos Friday and Saturday at the CSRG UNIX lab. Simple techniques for data entry, retrieval, updating and manipulation were shown on a rated list of Toronto restaurants.

An application being worked on using MRS is the FORM system, designed for handling office forms to be generated and dealt with interactively or automatically. Such operations as filling in a form, copying it, mailing it, discarding it, filing it, attaching it to a dossier and making an audit trail of it can be performed.

The FORM system is still under development. MRS is available for a $200 distribution charge from the MRS Distribution Manager, CSRG (address above). A copy of your UNIX or Mini-UNIX license and a signed University of Toronto Software Release Form must be provided. (See how important forms are?)

Speaker 28, at 9:55 **Mini-UNIX on the LSI-11**

Bob Hudyma,
Computer Systems Research Group
University of Toronto
Toronto, Canada M5S 1A1

Bob picked a slot in the session he was chairing to tell us how to adapt Mini-UNIX for the LSI-11 and up. The minimum requirements for the system are: the LSI-11; Extended Instruction Set Chip (optional but preferred); Serial Line Board DIJ-11; 28 Kwords of memory; and at least two AED or equivalent floppy disks (double density, DMA, 1200 512-byte blocks).

What you can end up with is a single-user system (it could accomodate 2-3 users performing small tasks with a fast disk like the RL01 or RK05); 18 Kwords of user-addressable memory (just enough to hold the C compiler); and a system that can handle practically all the usual UNIX software (nroff, yacc, V6 cc, ed, etc.). Forks are supported, and pipes are simulated via temporary files. The system can compile a 200-line C program to a.out in about 3 minutes. Its execution times compare favourably with LSX (which can, however, run programs about 4 Kwords larger).

Bob now presented the "Instant Do-It-Yourself Mini-UNIX-on-the-LSI Kit":

1) Write a device driver for the AED floppy disk.
2) Replace all references to PS in the system source by subroutine calls. Supply the appropriate routine in mch.s.
3) Eliminate clock references in main.c (these screwed up the LSI for some obscure reason).
4) Modify init.c for single-user only: remove references to checking the console switches.
5) Change the swap strategy to include a "swap flag" so as to prevent swapping while a swap is in progress on those slooooooow floppies.

If you send a copy of your Mini-UNIX license, the source for all this is available with the MRS distribution tape (see Speaker #27 above).

Speaker 29, at 10:15 **Real-time Data Collection and Failure Analysis**

Neil Groundwater
Analytic Disciplines, Inc.
8320 Old Courthouse Road, Suite 300
Vienna, Virginia 22180

Neil spoke about work he did when working for the New York Telephone Company on UNIX starting in 1972. An 11/20 was used for analysing failures in the telephone network with a view to locating areas with recurring problems (which would indicate some defect such as water getting into a phone cable).

Neil described the trouble reporting and analysis phases of the operation. Because of the quantities of data continuously being collected, it was necessary to filter out unimportant information. As a result, an "alerter" was written, which processed masses of data. The alerter would take note of certain problems in the system; any possibly critical problem would be sent to a monitored terminal immediately.

The system is now in use at N.Y. Telephone, collecting data from 96 simultaneous inputs. The data is manipulated by ATOM (Analysis Tool for Maintenance), a graph-oriented series of filters. The system runs now on N.Y. Tel.'s 11/70's., but is not available to those outside the Bell System.

coffee

Speaker 30, at 11:00 **How to get more out of your 11/70**

Dan Gielan
New York Telephone Company
375 Pearl Street
New York, NY 10038

Dan described N.Y. Tel.'s collection of 6 11/70's, each with 1 megabyte of memory and dial-in/dial-out facilities. Running USG-3 UNIX, they have hit a limit (per system) of 130 simultaneous processes, 300 open files, 320 inodes and 30 buffers. To gain more throughput they have implemented certain changes:
1) a swap device driver that lets the swap space size vary dynamically by overflowing onto another free device when the swap area is full;

2) checks for swapmap and coremap exceptions (so that when the system crashes you at least know why...);
3) the DH-11 driver and tty.c were changed so that in raw mode, the "kill" position in the byte table is used to denote a "message terminator";
4) RP06 drivers were changed by taking an RP04 driver and adding a search capability;
5) disk space assignment was improved;
6) mods were written to speed up the LP-11 driver
In design are a replacement for the RSW04, hanging memory off the MASS BUS; and a "smart" DH-11 driver with a Z80.

Dan had some recommendations as to what hardware should be obtained when funds are limited. In decreasing order of priority, they are:
1) 256 Kbytes of memory with FP11-C floating point box;
2) an independent swap device;
3) another 256 Kbytes of memory (Dan differs from the Bell people on this point);
4) separate controllers for the disks (or hang the TE-10 tape drive off the UNIBUS and free up one more of the 4 high-speed slots on the 11/70);
5) add a "smart" data handler (better multiplexer);
6) if you STILL have money to burn, get another CPU and split up the tasks.

Speaker 31, at 11:23 **Real-time data gathering on UNIX**

Eric Ostrum
Neurological Control Systems Lab
Department of Neurology
Carnegie Mellon University
5508 Walnut
Pittsburgh, Pennsylvania 15232

Eric Ostrum talked about the activities at NCSL in the field of research into brain functions and patterns and neurological diseases. UNIX is used for real-time data gathering of eye movements and gait patterns.

NCSL has developed the following packages: AD (Analog-to-Digital device driver); RA (read and interpret analysis data); PA (plot analysis data); EA (edit and manipulate the data). The system is available for $50 to educational users ($100 to commercial users) on a distribution tape. Separate I and D space is not required to run it.

Speaker 32, at 11:35 **Networking at Purdue**

Bill Croft
Electrical Engineering Dept.
Purdue University
West Lafayette, Indiana 47907

Bill Croft has developed a fairly complex networking facility at Purdue, where they have two 11/70's and two 11/45's. The 70's are connected to each other by DMC-11, as are the 45's. The links are about 1 megabaud bandwidth. A sustained end-to-end throughput of 250 Kbaud is about the same load on the CPU as a disk file copy.

The networking package features arbitrary interconnection: many individual connections are multiplexed through the same physical link rather than spooled. As a result,

a reasonable amount of interaction is available, with simple user commands for virtual terminal connection (*con*); remote process execution under a "connected shell" (*csh*); and file/directory transfers.

The user commands are all based on a function library which can be used by any C program to make one's own network connections. Functions exist for connecting, disconnecting, transferring files, reading, writing and signalling along the network. The net looks like a UNIX device driver: stty's are used to manipulate connections to other hosts. A connection is specified by 4 bytes (local-host, local-socket, foreign-host, foreign-socket: sockets are ARPA-style sockets). Many connections are allowed to the same host/socket pair as long as the 4-byte name is unique. This vastly simplifies the establishment of connections.

The whole package is available free as part of the Purdue distribution tape.

Speaker 33, at 11:48 **Networking at NYIT**

Bill Lindemann
Computer Graphics Lab
P.O. Box 170
Old Westbury, NY 11568

Bill spoke about the networking at NYIT, where an 11/35 is used as a front end for most of the terminals. Connections between machines are via DR-11C's which makes it seem as though there are serial lines. As Tom Duff mentioned (see Speaker #21), there are lots of machines to be networked.

Speaker 34, at 11:53 **RT/EMT: an RT-11 emulator on UNIX**

Mike Tilson
Human Computing Resources Corporation
10 St. Mary Street
Toronto, Canada M4Y 1P9
(416) 922-1937

Mike Tilson spoke about HCR's RT-11 emulator. This system actually does two separate things: run RT-11 binaries unchanged on UNIX; and run an RT-11 command interpreter within UNIX.

There is a need for such software because: (a) UNIX provides a good environment for the development of RT-11 software; (b) RT-11 applications programs can be brought up very quickly under UNIX; (c) there are always users who, for reasons of laziness or otherwise, do not want a change in their "operating system" at the surface level.

RT/EMT system overhead is comparable to RT-11. The features of the V3B SJ monitor are supported, and the system runs in user mode, so that minimal or no changes to the UNIX operating system are required. The RT-11 file system is simulated with UNIX directories containing ordinary UNIX files, so UNIX file manipulation programs may be used usefully.

The *rt* command will run any RT-11 binary, such as MACRO, LINK, LIBR, FORTRAN, PATCH, OMSI, PASCAL, EDIT, or TECO (with the V52 screen). The command *rtcom* puts

you into the RT-11 world with the exception that the keyboard-editing conventions of UNIX rather than of RT-11 are supported. (Since many UNIX installations now use the DEC conventions anyway, this is hardly a problem.)

The emulator works by loading itself into core in a large user area and copying itself to the top. It takes about 7 Kwords of memory in 28 Kwords of user space. (If thrashing is a problem, it can easily be reconfigured for a smaller memory size.) Other than the startup overhead, the emulator is comparable to actual RT-11 in speed.

RT/EMT is available with full source and documentation, along with a year's worth of bug reports and updates, for US$1350 to commercial users ($540 for each additional CPU); the price for non-profit educational institutions is US$810 ($315 for each additional CPU). There is a small additional shipping charge.

Speaker 35, at 12:20 **Software Tools Users Group Report**

Dennis Hall
Lawrence Berkeley Lab
University of California
Berkeley, California 94720

Dennis summarized the meeting of the Software Tools Users Group held on June 19, 1979 at the Westbury Hotel, Toronto. Of the 90 attendees, 40 were Software Tools types only and 50 were UNIX people as well. Dennis will be writing up the meeting in a forthcoming issue of *;login:*.

LUNCH

FRIDAY AFTERNOON
Session G: Improvements to the UNIX Operating System
Chair: Tom Duff, NYIT

Speaker 36, at 1:45 **Anyone doing system performance monitoring?**

Phil Poulos
Computer Systems Research Group
University of Toronto
Toronto, Canada M5S 1A1

Phil announced that he is doing some work on systems monitoring. He would like anyone who has done evaluation and/or prediction of UNIX system performance to contact him.

Speaker 37, at 1:40 **USENIX Committees**

Lou Katz (speaker #13) announced that committees have been formed with the following chairmen: Site committee (for next summer's meeting): Ben Woznick, BBN, Cambridge. Agenda of the Boulder meeting: John Donnelly, NCAR, Boulder. Tape distribution: Reidar Bornholdt, Columbia University, New York. The Nominations committee for the USENIX Board of Directors has not been formed yet.

Speaker 38, at 1:47 **11/40 Kernel Multiple Address Space**

Alfred Whaley
Department of Computer Science
University of Illinois U-C
Urbana, Illinois 61801

The UNIX installation at Urbana ran into a shortage of buffer space and lack of room for device table entries. The problem was caused by "too good I/O equipment": one fast powerful multiplexor which handles 128 DMA full-duplex channels, serial and parallel lines, printers, readers, and machine-machine communication.

The immediate goal was to put more code and data into the kernel than will normally fit. What was developed instead was a package of software which makes it possible to reconfigure the system easily with *any* space problem. There have been attempts at this sort of thing before, but MERT is "too much and too slow", and the Calgary buffer system lets one move only buffers out: when those are gone, then what?

The multiple addressing system developed works with separate "kernels", each with its own virtual addressing, text, data and BSS. Into "SYS" goes all the usual stuff from the sys directory (sleep, wakeup, system calls, scheduler, etc.); "B" devices are block devices such as disk drivers; "C" devices are character devices such as mem, tty's, and others; "BEGIN" contains the main() routine.

A few programs had to be changed to correspond to the kernel changes, notably *ld*, which now has a -v flag which combines with the "-o outfile" option to split up the kernel into its various pieces. A change to *cpp* (C preprocessor) gets around having to modify existing C source to run under the new kernel.

The system seems to have worked so far. It's available on tape from Alfred Whaley.

Speaker 39, at 2:05 **UNIX for 1100 Users???!!**

Ian Johnstone
Australian Graduate School of Management
University of New South Wales
P.O. Box 1
Kensington 2033, Australia

The University of New South Wales has been running UNIX for four years (since Version 6). They have 15 PDP-11's running UNIX. The particular site which Ian described is an 11/70 with: FP11-C floating point; 640 Kbytes of core; two 84 Mbyte UNIBUS disks (which are Ampex DM9100 and NOT recommended); eight DZ-11 multiplexors; 40 VDU's at 2400 baud; 10 DECwriters; and lots of line printers, card readers, tape drives and so

on. The system currently supports 1100 (yes, eleven hundred) users with 13,000 connect hours per month with general access only on weekdays. Between 7 a.m. and 8 p.m., at least 37 terminals are logged in 50% of the time; 43 terminals 25% of the time. Most of the workload consists of undergraduates using the Berkeley Pascal and EM.

To support this kind of use, two types of changes had to be made: those to accomodate a huge user population; and those to improve response with many simultaneous processes.

Changes to accomodate 1100 users include: a restructured passwd file, in binary rather than ASCII, with extra information stored for each entry; resource control (disk space quotas, process quotas, terminal restrictions, terminal booking, and proper accounting); dump and restor for a large file system; error logging in a file rather than the console; recovery from power fail and parity errors; *dcopy*, a disk compactor; and printf as a system call.

Changes to improve response include: an unrestricted number of buffers with faster buffer lookup via hashing and a pool of headers for raw I/O; a *fully* optimized DZ-11 driver; better process handling; disk driver optimizing; reduced swap activity; improved scheduling; and lots more.

A comparison between the virgin V6 UNIX and the current UNSW UNIX shows: maximum number of simultaneous users increased from 37 (with intolerable response times) to 44 (with satisfactory interaction); average time to compile and execute a 2-second Pascal program down from 158 seconds to 32; compile and link /unix (on an idle system) down from 10 minutes to 8. The first two results were obtained on a fully active system under normal operating conditions. System monitoring shows that the percentage of CPU time spent in "user" rather than "sys" mode has increased, on average, from 15% to 31%.

The entire UNSW system-improvements package is available on the conference distribution tape. The file /usr/sys/defines.h contains full information about the changes.

Speaker 40, at 2:30 **Another large UNIX system**

George Goble
Electrical Engineering Department
West Lafayette, Indiana 47907

George described the 11/70 at Purdue with a user population of 1400. Last year the system supported 63 at once at the worst time; this fall they expect 80-85 simultaneous users. With that amount of computing to be done, compiles and nroffs will probably be forked off to other machines with Bill Croft's networking (see Speaker #32). Because of the number of users, GID's are not used. The ka5 address is extended to fit in extra buffers, procedures and so on: between BSS and the kernel there is a new area, starting at virtual 120,000. This whole region is less than 64 Kwords. The system has a dual swap device, with a fast primary disk (RS04) and slow secondary swapping. Small programs are, of course, swapped on the fast disk.

There is a new system call, nicer(II), which will let one change the priority on another process. The set-gid bit is used (since there are no groups) as a "game" bit — the process runs at rock-bottom priority. There is also a "research" bit, for programs run in background mode which compute *only* when there is no interactive process running. Power failure recovery has been implemented. Automatic file system recovery is performed by *bootchk*, a program written in 1977 by Mike Tilson at the University of Toronto.

The Purdue people have had a lot of problems with their UNIBUS disks and recommend that one should never put a disk on the UNIBUS. Use the MASS BUS.

The entire collection of system changes, along with lots of other goodies (including the MIT dungeon) is available from George for free. Send him a tape and return postage if you want a copy of the Purdue distribution.

Speaker 41, at 2:50 **Office Use of UNIX**

Walter Lazear
Air Force Data Services Center
The Pentagon, Room 1D988
Washington, D.C. 20330
(202) 695-6101

The Air Force Data Services Center has four 11/70's for administrative support, handling text processing and similar tasks. The machines are up 22 hours a day, 7 days a week running practically nothing but nroff, troff and editing. Each 11/70 runs V6 with 40 simultaneous users. In addition, there is an 11/35 for the programmers to play with, which has compilers and so on. All machines are equipped with auto-dialers, so they can call each other or anyone else.

Walter described modifications to UNIX made at AFDSC. One such mod is the proper implementation of exit codes, including a specific code for "bad format". When the shell receives this exit code from a process, it looks up the correct usage in the manual section and informs the user. Furthermore, a consistent user interface has been developed for argument formats in all programs.

Other changes include: a read-only root file system; separate from the user file system; a system call telling you what machine you're running on; and a strategy for solving the problem of bad disk blocks, particularly in the swap area. The last 48 blocks of each file system are reserved as "alternates", and a table in the super block (the last 48 words) holds the bad block list for that file system. Mkfs has been modified to read and write every block in the file system, note all the bad blocks, and only then mount it. As a result, mkfs takes about 45 minutes to make a 56 Kbyte file system; however, once made, file systems run much better. The critical area is reduced to only two blocks (boot and the super block), and even programs which know about the internals of the file system (like df) will still work. Since installing the change, a total of 18 system-months have been logged on three systems without disk error. The overhead for the process is less than 1%.

The collection of AFDSC mods is available from Walter on a distribution tape.

coffee

Speaker 42, at 3:50 **Implementing C and UNIX more efficiently**

Carl D. Howe
Bolt, Beranek & Newman, Inc.
10 Moulton Street
Cambridge, Mass. 02139
(617) 491-1850 x3642

Carl described the environment at BBN, where they have 8 DEC-10's going full blast (and work for lots more). They have an 11/70 running UNIX with 150 users. There is, however, a demand for a machine with the performance of the 11/70 (running UNIX) at much less cost.

The BBN Computer Company (a subsidiary of BBN) has designed the Microprogrammable Building Block (MBB), a fast microprogrammable processor to be used as the basis for special purpose computer systems. This processor features a 135 nanosecond microcycle time and up to 16K of 32-bit microcode memory words. The machine also has 1024 fast hardware registers. Its architecture makes emulation of already-existing instruction sets easy and efficient. Furthermore, it is inexpensive to build and easy to service.

The MBB has been built and is being used for other projects. It emulates the Honeywell 316 one and one-half times *faster* than the 316 runs its own code.

The MBB has a *macrocode* which is distinct from its microcode. The macrocode is an intermediate code produced by the compiler. It is *not* designed as a users' assembler and hence is often ugly but efficient. The objective was to optimize the instruction set in the direction of efficiency at the expense of aesthetics.

In constructing a C compiler for the MBB it was desired to avoid some of the problems with the PDP-11, such as the 16-bit address limit, the limit of three hardware registers available to the user, the slow speed of subroutine calls and the inefficiency of local variable addressing. These problems have, it seems, all been solved. With 20-bit words, larger processes may be run on the machine. The abundance of hardware registers makes register saving unnecessary (except with large amounts of recursion, in which case 64 are saved at once). The macrocode can call the microcode to do certain things; there is flexibility in deciding what will be implemented in microcode. And the microcode has an instruction set with many different addressing modes, so that doubles and longs can be dealt with directly.

The assembler for the MBB was written in 3 days (using yacc). The compiler is being written and compiled with the phototypesetter-version of cc. The microcode has been written and the hardware already exists although it is not yet generally available.

Once C is working, BBN will begin implementing UNIX on the MBB. UNIX will feature: a 20-bit MBB processor; 128K 20-bit words of memory; two disk drives of 40 to 300 megabytes; one 8-line serial multiplexor for terminals; and a compute network interface. It is expected to be running by the end of the summer; disk interfaces and memory management have not been designed yet. While nothing can be promised as yet, it is expected that BBN will be marketing the MBB under UNIX by the summer of 1980, and that it will give the performance of an 11/70 for the cost of an 11/34.

Speaker 43, at 4:10 **UNIX on a UNIVAC V77-600**

Harold Pierson
RLG Associates, Inc.
11250 Roger Bacon Drive, Suite 16
Reston, Virginia 22090
(703) 471-1108

Harold Pierson described UNIX v77 on the Univac as "an experience in portability". RLG has been doing the project for Sperry Univac, who took over Varian and are producing the "Varian V77" as the "Univac V77". The writing of the C compiler was subcontracted to Cassandra Inc. (Bob McClure). UNIX is now up and running most of PWB on the V77.

Harold discussed some of the problems resulting from the fact that the V77 is a word-addressed machine. Assignment conversions, parameter passing and word/byte relocation in the loader all presented a problem. The solution implemented was to pass always by byte pointer when calling subroutines. Within routines, word pointers are used. The loader was modified to cope with this.

The were other assorted incompatibilities. The final result was code that is larger by 20-30% on a machine which is much more powerful and faster than the PDP-11. Additionally, because of 512-word paging, there are 16 address spaces in core; instead of just the kernel, the supervisor and the user you can actually have 15 users running in core.

At present UNIX (the real thing, not a look-alike) is running multi-user on the V77-600. RLG is handling the program development and administrative software; Sperry Univac will be marketing it in the future. Other developments to follow are a version for the v77-800 (with separate I and D space), multiple address spaces, scatter page swapping and micro-coded assists.

Speaker 44, at 4:35 **An Accounting System for UNIX**

Robert N. Jesse
Johns Hopkins University
Electrical Engineering Dept.
Barton Hall
Baltimore, Maryland 21218

The 11/45 at JHU is being used by students, which automatically creates a hostile environment. All the things that one can normally get away with on UNIX must be monitored and controlled.

The /etc/passwd file has been left intact, but a parallel file, /etc/uif, contains more information for each user: the encrypted password; time and tty of last login; cpu and connect time used and quotas; block-I/O and line-printer-page count and quotas; login terminal restrictions; and special permissions and options.

Monitoring of system usage is done by a new program, *init2*, which receives the termination statuses of all programs run by the used on a pipe from *init*. The information is passed with a new system call, waitinfo(), which provides the proc and user structures of the deceased process. When the shell exits, init calls *quota*, which will not let the user log out until he has reduced his disk consumption below his quota. (If the user is on a phone line and hangs up, he will not be permitted to log in again by phone until he

has discussed his problem with the system administrators.) Another system call written for init2 is getppid(II), which returns the PID of the parent. The program *gcheck* is run periodically to check for people owning files in others' directories.

When the distribution of this system is ready, it will be available to educational UNIX licensees only. Those wishing to obtain it must sign a non-distribution agreement (the JHU people want all copies to come from them).

Speaker 45, at 4:55 **A High Performance UNIX System**

Mike Muuss
Johns Hopkins University
Electrical Engineering Department
Barton Hall
Baltimore, Md. 21218

Because it was late, Mike spoke very briefly about the changes made at JHU to improve performance. "Performance" in this context is not just good response time; it includes:

(a) excellent response to small interactive processes, while keeping the CPU as full utilized as possible;
(b) good security in a hostile user environment;
(c) detection and avoidance of certain hardware failures;
(d) good data integrity, especially in a hostile hardware environment;
(e) a more pleasant user interface; and
(f) accountability and restrictions on resource consumption.

The JHU people have divided the solution up into four areas: (1) resource measurement and accounting; (2) teletype driver; (3) security and integrity; and (4) performance mods.

Resource measurement. See Robert Jesse's talk (Speaker #44) for a description of the accounting system. The file /dev/data contains dynamically modified information about processes running, system load and performance, and resources.

Teletype driver. Lots of enhancements increasing performance; others adding a better user interface: ctrl-r to retype the buffer; page mode; stall mode; character rubout via BS-SP-BS; ctrl-t to print status of process being waited for.

Security and integrity. Special files and SET[UG]ID operate only on the root file system; front panel interference protection (!); /dev/error and /etc/logerr for logging and recording history of non-fatal errors, along with interpretative software; memory test and lockout at boot time and dynamic memory testing; a stack limit register; use of IDB - kernel runs a true split I and D space, with the I-space write protected; fixing of assorted errors and bugs in the system source.

Performance mods. Process dispatching has been fixed up by implementing full process queues. The result has been much faster reponse time for editors and other small CPU usage programs. (½-second typical reponse, with 10-15 users on the 11/45). Swapping increases, however, so it is advisable to have an RK05 dedicated to swapping. In the future, I/O will be integrated into the dispatching priority. The priority evaluation done now keeps track of how long the process was in core before and takes into account the size of the process.

Some fatal bugs in the system source were also found, but Mike could not discuss them publicly because of the presence of "aliens" (people not bound by a Western Electric non-disclosure agreement).

Speaker 46, at 5:15 **UNIX on the IBM 370**

Steve Bellovin
University of North Carolina
New West Hall 035 A
Chapel Hill, North Carolina 27514

Steve spoke about the Triangle Universities Computing Center (TUCC) group, which has been running TSO for six years and would now like to move to a time-sharing system. The plan is to move the command level of UNIX to an IBM S/370-165/II, and thence to the Amdahl 470. The reason these machines have been chosen is that their hardware and hardware service are felt to be far more reliable than DEC.

The 370 UNIX will be CRT-terminal oriented. It will be an EBCDIC system (ugh!). Assorted-stuff will be imported from other systems. It will be implemented using VM-370; the goal is to have 100 simultaneous users, each with a virtual machine. The UNIX file system, pipes and shell will be kept intact.

It is hoped that in a year's time the system will be operational and capable of handling 20-30 simultaneous users. By two years from now it should be on the market. Steve has no idea yet as to what the price will be, or whether it will be available cheaply to educational institutions.

Does anyone have a DX-11 driver for Steve?

SATURDAY MORNING
Session 7: Educational uses, and what's happening at ?....
Chair: Sandra Wright, DCIEM

Attendance was low at Saturday's session (about 35 instead of 350). These notes were taken by Sandy Wright and expanded by Dave Sherman.

Speaker 47, at 9:05 **UNIX in the Undergraduate Lab Environment**

Don Schertz
Bradley University
Dept. of Electrical Engineering
Peoria, Illinois 61625

UNIX at Bradley is running on an 11/40, where half of the connect time is spent on software development for a microprocessor. RJE into the university's CYBER is planned in the near future. They are running Mini-UNIX from the Johns Hopkins University, which is a good package. Communications to micros is being worked on. Currently RS-232 loops are used; but there will be a move to the 8080 on parallel ports. With a variety of cross assemblers available, the micros are downloaded with binaries.

Speaker 48, at 9:15

UNIX in a large educational environment

Mike O'Dell
University of Oklahoma
Engineering Computer Network
Engineering Center
Norman, Oklahoma 73019

The University of Oklahoma's 11/70 has 384K of core with a floating-point processor and RM03's on separate controllers; four DH-11's (8 dialup lines), and a user population of 500 to 700. The maximum number of users supported simultaneously is 37. They use the Purdue tty driver with some mods for weird terminals.

Work is being done on RJE to the IBM 370/158; they are trying to get the PWB RJE software working.

Speaker 49, at 9:30

QED, or The Little Ed That Grew

Robert Pike
Computer Systems Research Group
University of Toronto
Toronto, Canada M5S 1A1

David Tilbrook
Bell-Northern Systems Research
247 Brunswick Avenue
Toronto, Canada M5S 2M6

CSRG at the University of Toronto has had *ed* hacks for a long time. Years ago, Tom Duff and Rob Pike added assorted useful features; these were expanded and rewritten recently by Hugh Redelmeier. Ed at U of T now has the following features: a single error character following the '?' indicating what type of error was found; '*' as an easier way of typing 1,$ (or N* for N,$); '&' as a simple form of page-oriented addressing; file saving upon a hangup signal (in "filename:sav"); extensions to regular expressions for easier matching; s2/xxx/yyy/ for the second occurrence in a line (and so on); 'cd file' for changing the working directory; 'x' command for character editing; 'j' to join lines; 'u' to undo the last substitution; and others.

Qed was originally written by Tom Duff by taking the code for ed and adding multiple buffers, number registers and assorted other goodies. Improvements by Hugh Redelmeier, David Tilbrook and Rob Pike have taken place over the intervening four years. Although more enhancements are still in progress, qed is already a powerful editing tool.

The principal features of qed are multiple file/multiple buffer editing, macro capability and an enhanced interface to the shell. For experienced programmers, these features can greatly simplify difficult or repetitive text editing tasks, and make it easy to develop special-purpose editors that have knowledge about the structure of the text they are working with.

As much as possible, qed is a rigorous superset of the version of ed running at U of T. The couple of minor differences are considered to be improvements over the original ed, but it was decided to leave ed's behaviour as it was to avoid incompatibilities with eds running elsewhere (specifically the PWB (Bench) version).

The main extension in qed is the presence of 58 buffers, labeled by upper and lower case alphabetics and four other characters which are reserved for several purposes. All normal ed commands work within a buffer, so that internal to a particular buffer the user sees what appears to be a regular editor. The only exceptions to this are the move and copy commands, which allow inter-buffer transfers, and, of course, the command to change the current buffer.

At any time when input is expected, be it command or append input, the contents of a buffer may be inserted using a simple escape sequence. As well, qed provides registers which are accessed in a similar manner, but which have some commands to deal with them directly rather than by changing the current buffer. The user, at startup, can specify the name of a file which is to be read into a reserved buffer and executed before reading commands from the terminal, so that he can preset registers and buffers to contain useful commands and text.

For programming purposes, qed also has simple control structures and message printing capabilities, so that it can be used as a rather simple programming language. The implementers of qed claim that their version is considerably better human engineered than previous versions of qed on which it is based.

A tape with source and documentation for qed, U of T's ed, and a few other useful programs, may be obtained from ~~David Tilbrook~~ Rob Pike. Please send a blank tape, address label and return postage.

Robert Pike
Caltech 220-47
Pasadena CA 91125
213-795-6811 x 263
or x 1245

coffee

Speaker 50, at 10:50

News from the U.K.

R.P.A. Collinson
University of Kent
Canterbury CT2 7NF
Kent, England

The are 15-20 UNIX sites in the United Kingdom, all with educational licenses. Most of them are 11/40's, although there are three 45's and one 70. They are in touch with European UNIX sites in Holland and France. Only Glasgow has tape drives – RK05's are used for the most part, which has the side effect of making distributions more selective and carefully thought out.

As there are only perhaps five system hackers in the whole of Great Britain, most locations stick to the conventional stuff, using "Standard C" (whatever that is), and keeping any changes to system source code well documented.

Languages being developed in Britain include: BCPL (two different versions of this portable high-level predecessor to C); Modula (at the University of York: Ian Cottam); PDP11 (Sussex: Steve Hardy); and micro assemblers in various places. An implementation of Algol 68 – a major project – is underway at St. Andrews.

The UK UNIX User Group has been in operation for two years. It is a DEC SIG; it publishes a (theoretically) quarterly newsletter. Glasgow acts as a software distribution centre. The users' meetings draw about 50 people. UK/UUG is prepared to exchange tapes with USENIX.

Speaker 51, at 11:05 **What's happening at BBN**

Carl D. Howe
Bolt, Beranek & Newman, Inc.
10 Moulton Street
Cambridge, Mass. 02139
(617) 491-1850 x3642

Carl briefly mentioned a number of the projects underway at BBN: the new C machine (see speaker #42); a C compiler for the Z8000 and the DEC-10; manual revisions; extended documentation: non-blocking I/O; ports, networking; a new Rand editor – for CRT's with cursor addressing and shared text; word operations user configurable; cross-net debugging; screen managers under investigation; ARPAnet work; work on high bandwidth local networking is starting.

The BBN UNIX mods distribution tape costs $300.

Speaker 52, at 11:15 **Tuning PWB UNIX**

George Pajari
GTE Automatic Electric (Canada) ltd.
100 Strowger Blvd.
Brockville, Ontario, Canada K6V 5W8

The RJE modification under PWB needs rewriting. Kernel modifications have also been made at GTE: a query feature which gives utilization of tables; and *gmon*, which keeps track of high- and low-water marks in tables. Knowledge about these watermarks allows sensible settings of table sizes, and also prevents some crashes by detecting table overflows.

This software might be available. Contact Paul Hart at GTE (address above).

Speaker 53, at 11:20 **Screen editors**

Mark Pearson
Yourdon Inc.
1133 Avenue of the Americas
New York, NY 10036

Yourdon has a screen editor which works as a front end to *ed*. It supports dumb terminals. The backslash character is not special. The cursor can be moved a word at a time. It should be able to front onto the Toronto qed, and it was agreed by Dave Tilbrook and Mark Pearson to try and do so.

VAX VMS/UNIX COMPARISON

Carnegie-Mellon University will probably be getting some Vaxes (Vaxen?) for general timesharing use in the CS departent, and have to decided whether to go for VMS or UNIX (possibly modified), or to build a completely new operating system. There is a long sequence of comments and suggestions, mainly comparing VMS and UNIX, in the files vax.msg[c380sf50], vax1 and vax2. I have selected the message giving the initial comparisons, and some remarks on Unix from Bill Joy of UC Berkeley. Most of the other 40-odd messages are rather dull statements of opinion, but there is some interesting discussion of RMS, the VMS file-system. The question is whether you want some system-provided structure for files, or whether this should always be the preserve of user-level programs (though possibly hidden in libraries).

----Message 2 is----
Date: 19 April 1979
From: McKeown at CMUA
Subject: VMS/UNIX Comparison

SOME COMPARISONS ON VMS AND UNIX FOR THE VAX/780

James Gosling
Steve Shafer
Dave McKeown
April 19, 1979

This is intended to be a first pass at comparing features and functions available in VAX/VMS and VAX/UNIX. We expect to expand this document, to flesh out some of the statements, and add to or modify our analysis as we receive comments.

All comments are welcome!!!

Vax/VMS features	VAX UNIX features
- quotas to restrict resource usage no one user can monopolize resources	- no quotas for anything.
- more redundancy in the file system [as reliable as tops-10]	- fragile file system, but there are many tools for crash recovery, this requires wizardry. Overall, more fragile.
- has a more complete set of file access protection modes	
- has a TOPS-10 like search list	- search lists are wired into individual programs
- has a generalized asynchronous IO facility.	- all IO is synchronous.
- has a generalized demand-paging virtual memory system. There have been uncomplimentary comments about	- a fairly traditional swapping system.

he scheduler.	
ible to map files into a users ddress space.	
nterprocess communication is more laborate. has named mailboxes.	- interprocess communication limited to simple pipes; both ends of which must be inherited.
upports DECnet	- supports the Bell internal net; code for ARPAnet support exists.
as a reasonable program sharing acility; the library sharing acility is quite deficient though.	- single segment program sharing; no subroutine library sharing. (I mean sharing in the use-the-same-physical-copy sense)
he 'system' language is BLISS-32 he OS is written almost entirely in ssembler & is hence very difficult o modify. The mere mass of the ode makes it difficult to nderstand.	- the 'system' language is C - the OS is written almost entirely in C which makes it very easy to modify. The entire Unix kernel is very small; clean and easy to understand. Relative novices can work in the bowels of the OS with little time spent learning how it it works.
ernel calls are baroque & difficult o use; they use complicated control locks and calling sequences. he command interpreter is quite onventional.	- there are very few kernel calls; all taking a small number of simple arguments. - the command interpreter is quite sophisticated; it has facilities to interconnect programs via pipes and execute them in the background in a natural manner.
ild card file names are special ased on a program-by-program basis	- wild card conventions are uniformly applied by the command interpreter.
ery little existing software; ssentially just that from DEC.	- there is a vast body of existing software for Unix; most of which is trivially transportable to VAX. [notably: many editors; compilers; compiler generators; document production programs; the list is long]
he manual is distinct from the on-ine help. Many feel that the anual is overly verbose. Copies of he multi-volume manual must be btained from DEC.	- the manual and the on-line help are one and the same. It is quite concise & generally clear. This arises primarily from tight standards & a clean system. Since the manual is online; we can produce; and have produced; copies locally.
he file system is hierarchical with any different file access methods; ut this too has a baroque and hard-o-use flavor.	- the file system is hierarchical only one file access method: random access byte stream; it is simple and easy to use.

hing that one can do in Unix one can do in VMS; while the verse is not true; there are things which can be done in VMS h cannot be done in Unix. But there is a very clear tendancy for VMS to be very intricate and hard to understand; it has many features that result from a desire to be compatible with other systems and from an IBM-like desire to be all things to all customers. In many cases the same feature is repeated; but in a slightly different flavor. On the other hand; Unix is extraordinarily simple with a small number of orthogonal features. It does not attempt to cover everything that everyone would like to do; it has sacrificed functionality for simplicity.

The point about the language in which the OS kernel is implemented is significant. The fact that the Unix kernel is small & written in a high level language means that it is easy to understand and modify; while on the other hand VMS is written in assembler; which along with the complexity of the OS would make it difficult to either understand or modify. Many of the features that are missing from Unix are essentially trivial to add. Witness the immense number of mods that have been made to the IUS and SUS Unix systems.

----Message 11 is----
Date: 5 May 1979
From: WNJ at MIT-AI (William N. Joy; U. C. Berkeley)
Subject: VAX UNIX

You should really talk to us folks here at Berkeley about VAX operating systems. We have experience with both VMS and VAX/UNIX (which we were the first to run outside of Bell Labs). We are putting VM into vax/unix and have a lisp system; screen editor; etc already running.

There are a number of misconceptions in the file you have on VAX Systems. To wit:

1. The new version of UNIX does not have a fragile file system. We never have any trouble.

2. Search lists are not wired in; rather are part of the ENVIRONMENT (a new concept in version 7 UNIX).

3. A paged version of vax/unix will be running here with simple sharing soon. The form of desirable sharing in a vax-like architecture is under discussion; VMS certainly doesn't have the answer.

4. There is a decnet driver for UNIX readily available (written byy Jim Hamill of DEC). We have a version running between an 11/70 and a vax (running vms) here over KMC-11's.

5. VMS drivers are difficult to add and a mess; as is the entire system. We have added several drivers to UNIX easily. Looking at a VMS driver is a waste of time.

VAX/VMS "wish list"

"able to leap tall terminals in a single bound"

"more powerful than a speeding abacus"

"if you can imagine a wombat leaping in and out of
a creek 10,000,000 times then you get some idea
of our useful it is"

USERNAMES, LOGGED-IN DIRECTORIES, and ACCOUNTS.

We require

(1) INDIVIDUAL USERNAMES. No more than one person per username, so that files created by one person can be protected from deletion by another.

(2) SEPARATE ACCOUNTS. No more than one course or research activity per account.

(3) INDIVIDUAL DIRECTORIES. For example, a student doing computing in several courses requires a separate directory for each course.

(4) PRINTOUT IDENTIFICATION BY USER rather than by course or research activity.

(5) MNEMONIC usernames, accounts, and logged-in directories.

Therefore, we would like to have

USERNAMES LONGER THAN 12 CHARACTERS

PERIODS (subdirectory-style) ALLOWED IN USERNAMES

ACCOUNTS LONGER THAN 8 CHARACTERS

PRINTOUT IDENTIFICATION BY SUBFIELD OF USERNAME

Explanation:

It seems that each user must have a separate username for each course and research activity. We have many courses, and in a course there may be several students with similar names. 12 characters per username is barely adequate. The university uses 6-character codes to identify courses, and we would like to use them in usernames and accounts. The university uses 8-digit numbers to identify staff and students; we would prefer to use users' names in usernames.

The limit of 9 characters per filename hinders us in giving meaningful logged-in directory names corresponding to usernames. Using second-level directories as logged-in directories is convenient and gives us longer identifiers, but usernames can't be the same because periods are not allowed (the login procedure rejects them, despite allowing $_%+-).

Usernames should be able to contain: course code (6 characters), separator (1), and user's name (say 20). Accounts should be able to contain: school code (4 characters), research/teaching indicator (1), school subdivision code (4), course code (6), staff/student indicator (1), and preferably staff/student number (8) and field separators (4 or 5).

Examples:

USERNAME: MAPH102.FITZHARDINGE,MICHAEL

ACCOUNT: MAPH_T_COMP_MAPH102_C_77071263

(School of Maths & Physics, teaching, computing, intro. to computers, student, student number)

We want printouts for this user identified "FITZHARDINGE", NOT "MAPH102..." or "EX1.FOR", so that Fitzhardinge might be able to retrieve it before it is taken by another student in MAPH102 who has a program called EX1.FOR - or, more importantly, doesn't have one yet and would like to copy somebody else's! Operators put printouts in boxes according to the first block

letter on the header page. Putting all "MAPH102" or "EX1" printouts in the same box makes life difficult for honest users and easy for cheats, and will lead to the M or E boxes becoming very full: users' names are much better distributed over the alphabet than are course codes or file names.

FILE ADMINISTRATION

BACKUP, RETENTION PERIODS, VERSION RETENTION LIMITS,
TEMPORARY FILES, DISK QUOTAS

BACKUP: BCK: doesn't work. can't handle subdirectories properly

DSC: takes too long; makes it too easy for operator to destroy vital information

DISK QUOTAS: Submanager facility or distributed control desirable. We want to be able to carve up the file store into schools, then let the individual users in the school fight it out within the school without having to bother us.

DISK QUOTAS must be INSTANTANEOUS in effect: we have already had a problem with a student program with an 'output' statement in an infinite loop eating up all available disk storage on a device!

ELSE: In case you should ask "Why does Macquarie need temporary files, etc., once it has disk quotas" :-

Consider a class of 500 students being introduced to Fortran programming. They require the ability to save files for future use. What happens is that each student quickly reaches his disk quota, and continues using his disk quota until the course is over, but in fact much of his file usage is unnecessary: he won't delete the files from his first assignment until he has to start on the next (a few weeks later); not understanding the significance of files such as EX1·OBJ and SOSCOO·TMP, he is inclined to leave them alone and not delete them (especially if they're locked!). Much disk space would be saved by assigning the student:-

VERSION RETENTION LIMIT : 1

ALL FILES TEMPORARY UNLESS EXPLICITLY SAVED

RETENTION PERIOD (for SAVED files) : 14 days

FILE ACCESS

CLASSES

We require, in order of decreasing priority:-

(1) SHARED READ-ONLY FILES that students can read and execute only.

(2) PROTECTION of everyone's files from being modified or deleted by students.

(3) PRIVACY among students: students should not be able to read or discover the names of other students' files.

(4) STAFF ACCESS to students' files: staff should have full access to files of their students.

(5) ENFORCED PRIVACY: there should be nothing a student can do to give students more access to his files.

(6) TEAMWORK: the staff member in charge should be able to establish teams. Students should have read and execute access to the files of fellow team members, but not to the files of other students.

(1), (2), and (3) we can achieve by giving everyone an individual UIC. However, the restriction of 256 members per group means that some of our classes would require more than one group, which makes life messy. A solution would be to allow MORE THAN 256 MEMBERS PER GROUP.

(4) conflicts with (2) and (3) at present because all members of a group are considered equal. A solution would be for
GROUP PRIVILEGE TO GIVE OWNER ACCESS TO ALL FILES IN THE GROUP.
Another solution (not as good) would be for group members with member numbers less than 20 (say) to automatically have owner access. Another approach is to define GROUPS in the sense of TOPS-20 (see notes on (6)).

(5) is impracticable with the existing facilities: at present,
(a) A student can always change the protection on his files.
(b) A student can always change the protection on his directory, unless it has a different UIC. If it has a different UIC, then because the student needs read and write access to it, GROUP or WORLD must be allowed read and write access to it. This means that other students have read and write access to it, unless there is a separate group for each student!

A solution would be to allow the creation of directories whose protection cannot be changed by a user with the same UIC as the directory; some ideas:

(a) CREATE/DIRECTORY/LOCK [COMP1_SMITHJB]/OWNER-UIC=[314,234]
/LOCK means that the only users who can set the protection for this directory are those with the same UIC as the user who issued the CREATE/DIRECTORY

(b) 1. CREATE/DIRECTORY/SYSTEM [COMP1]/OWNER_UIC=[314,000]
2. CREATE/DIRECTORY/GROUP [COMP1.SMITHJB]/OWNER_UIC=[314,234]

/SYSTEM means that a system UIC is required to change protection.

/GROUP means that Group privilege is required to change protection.

(c) CREATE/DIRECTORY/KEY=BLAH [COMP1.SMITHJB]/OWNER_UIC=[314,234]
/KEY=key means that the given key must be provided on a SET PROTECTION command for this directory.

A nice generalization of (a), (b), or (c) would be to extend it to files in general.

(6) is impracticable with the existing facilities: there would have to be a separate group for each team, making life messy and requirements (1) and (4) difficult to fulfil.

A solution would be to provide a facility similar to GROUPS in TOPS-20.

(See TOPS-20 System Manager's Guide, Order No AA-4169C-TM, May 77, Section 5.5.)

A GENERAL SOLUTION

Base protection on Usernames rather than UICs.

Examples:

(a) SET PROTECTION/DEFAULT=(READ:STO1.*,WRITE:,EXE:*,DEL:-)

Gives read access to all usernames starting STO1,
write access to the current user and system users only
execute access to all users
delete access to system users only

(b) DEFINE "DAMPNEY,PAYNE,SMITH_B,PARCELL" STAFF

SET PROT=(R:-<STAFF>,W:-<STAFF>,EXE:-*,DEL:)/DEF

If this is the protection in force for COMP1.BLOGGS then:-

read and write access is given to COMP1.BLOGGS, COMP1.DAMPNEY, COMP1.PAYNE, COMP1.SMITH_B, and COMP1.PARCELL;
Execute access is given to all usernames starting COMP1;
delete access is given to COMP1.BLOGGS
and in addition, all accesses are allowed to system users.

NOTE: All that needs to be stored in the file attributes is the protection specification - not a complete list of all usernames granted each access!

USERNAME TABLE

READ FIELD	⟶	-<STAFF>
WRITE FIELD	⟶	-<STAFF>
EXECUTE FIELD	⟶	-*
DELETE FIELD	⟶	

Things so easy to implement, or so important, that we've done them ourselves.

LOGIN	Message of the day
LOGIN	Time of login
LOGIN	Purge logged-in directory
LOGIN	Site identification (MACQUARIE UNIVERSITY)
LOGIN	Check disk quota (we're only bluffing at present!)
LOGOUT	Site-specific LOGOUT.COM
LOGOUT	DELETE/CONFIRM/SINCE=login-time [logged-in directory]*.*;*
AUTHORIZE	Disk Quotas (we're only bluffing at present)
AUTHORIZE	User-specific default file protection (done by LGICMD)
LIBRARY	Site executable library, and easy way to access (e.g. @RUN MINITAB or RUN S:MINITAB)
LIBRARY	Site command procedure library, and easy way to access (e.g. @PROC COPYDIR, or @P:COPYDIR)
MANAGEMENT	Distinctive usernames for security-sensitive users (username starting with 7)
MANAGEMENT	Normal users have second-level, not first-level, directories
MANAGEMENT	User files on VMS disk (we just assumed this was OK, until a disaster wrought by VMSKITCPY caused us to start talking about it, and local DEC software engineer suggested we should not have this)
FILES	Recover lost files into user directories (command procedures written in the small hours after losing over 1000 files and finding VFY2 refused to put them anywhere)
HELP	improved descriptions of REQUEST
STARTUP	don't allow users on immediately (our solution is very crude: SUBMIT a procedure that WAITs 5 mins before doing a SET LOGINS)
OPERATOR	Operator has automatic REPLY/ENABLE on LOGIN & REPLY/DISABLE on LOGOUT.
OPERATOR	Command procedures to throw off all users or all users with group numbers higher than some limit. (@ALL OFF & @STUDOFF)
AUTHORIZE	File version retention limit (of 1, enforced by PURGE on LOGIN & LOGOUT)
CREATE/DIRECTORY	Allow defaulting, e.g. CRE/DIR [.MYSUBDIR] (@CREDIR)
DCL	User's own username available as symbol or lexical function
DCL	Time of login available as symbol or lexical function

PRIORITY 0·4

Things so easy to implement, or so important, that we're doing them ourselves at present.

BACKUP	Command procedures for foolproof regular backups
PRINT	Identify printouts by subfield of username, instead of by file-id. (See separate sheet)
AUTHORIZE	Produce a certificate of registration for user (showing username, UIC, directory, disk quota, and any characteristics different from DEFAULT's)
MOUNT	Send message to operater and wait for reply (we like Univac's @ASG for tapes)
COPY	Copy subdirectories [the files listed in subdirectory files]
ERRLOG	Automatically every day start a new error log and ~~purt~~ Print a roll-up report of the previous day's.

PRIORITY 0·6

Things that are so vital to us that we are about to do them ourselves.

BILLING/USAGE	Bill accounts, showing usage by username; provide breakdown of usage by school and type of work.
AUTHORIZE	Provide means for conveniently authorizing thousands of individual usernames, each with individual directory.
AUTHORIZE	Provide means for making global changes to users' quotas. [A "UAF editor"] (we have already had to change ASTLIM for all users; this will be difficult when we have thousands of users.)
TERMINAL UTILIZATION	Report on utilization of terminals.
TEMPORARY FILES	The files of certain users are liable to be deleted overnight unless they do something to save them. (see separate sheet)
GROUP LOGIN·COM	Performed compulsorily for all members of the group, after the System LOGIN·COM (which we've implemented) and before any user LOGIN·COM.

PRIORITY 1

Things that seriously detract from our ability to run the system in the way we want, but are too hard for us to work around.

VMS — Dynamic adjustment of working sets of current processes for maximum efficiency (fastest response? maximum throughput? Envisaged as the software equivalent of a knob on the console to expand or contract all current working-sets in proportion, with a meter showing efficiency).

DCL — Redefinition of existing DCL commands (we've attempted this for LOGOUT [to do PURGE, DELETE/CONFIRM/SINCE] and BASIC [to allow choice of BASIC-11 or BASIC-PLUS-2], but our users defeat our attempts by doing

```
$ LOGOX
or  $ LOGOUT := LOGOUT
or  $ DELETE/SYMBOL/GLOBAL/ALL)
```

DCL — Permanent symbols (proposed solution to problem above)

MANAGEMENT — Provide an easy, foolproof, standard way to maintain all software, including possible local changes in files or additional files (e.g. [SYSMGR]SYSTARTUP.COM, [SYSHLP] files), which allows economical use of disk space in the running system (i.e. disk on which software resides contains user files as well, and contains no unnecessary files)

BACKUP — Fast disk copy (a block-for-block copy to produce a backup copy for emergency use, as opposed to a DSC-merged copy)

BACKUP — Incremental backups (I.e. backing-up of only those files changed since last backup. BCK can't handle subdirectories properly).

AUTHORIZE — File retention periods, based on time since file was last accessed [e.g. executed] (preferably username-specific and capable of being over-ridden, like protection is supposed to be). (See separate sheet)

AUTHORIZE — Disk quotas.

AUTHORIZE — Connect-time quotas (preferably both accumulated time and time for any one session).

new

AUTHORIZE — Sub-manager facility, to allow each school's computing liason officer to authorize (how) groups in that school, and each group's owner to authorize new members of that group (we may well end up doing this ourselves, depending on how hard it turns out to be).

AUTHORIZE / FILES-11 — Improvements in Usernames, logged-in directories, accounts, and protection - see separate sheets:-

- Usernames longer than 12 characters (say 32)
- More than 256 members per group (say 16 bits)
- Group privilege to give Owner access within group

DCL — Standard method for sites to add local commands (e.g. MQU <COMMAND> cf MCR <COMMAND>); commands must be capable of being implemented by command procedures as well as executable images.

DCL — Addition lexical functions:

- parse a file specification, subject to given defaults (useful for problem above)
- given a file specification, return the full file identifier of the file referred to (as if about to OPEN a file)

DCL — Ability to suppress messages in command procedures (as in UNIVAC CTS)

(e.g. SET [NO]INFORMATIVE-MESSAGES [default SET NOINF...]
SET [NO]WARNING-MESSAGES [default SET NOWARN...]
SET [NO]ERROR-MESSAGES [default SET ERROR...])

AUTHORIZE — Require privilege to SET NOCONTROL_Y.
(we have had 3 student terminals hung in DCL loops with SET NOCONTROL_Y in as many days!)

TERMINALS — DO NOT require any privilege to SET TERMINAL /NOECHO or /PASSALL

DISPLAY — Allow user simple means of modifying DISPLAY to run on terminals other than VT52/5 (we have mainly VC404s, one ADM-3a)

REPLY — REPLY/USERNAME= , REPLY/ID= , REPLY/PROCESS=
(At present, we have no way of ~~replying to~~ SENDING unsolicited messages!)

PRIORITY 2.

VMS

Increase general consistency, flexibility, & commodity. An example of what not to do is the 'explanation' in VMS 1.5 release notes of how to implement a 'foreign' command; use a program. Now a program cannot invoke DCL (except by complicated fiddle using subprocesses), whereas DCL can invoke a program (by RUN). Therefore, if 'foreign' commands can be implemented one way only, it should be by DCL, not program. Of course, better still would be to (a) provide a good interface to DCL from programs, and (b) allow 'foreign' commands to use both methods.

Some suggested guidelines:

1. Anything that can be done by processing the output from a DCL command should be available as a lexical function or symbol, and from Fortran.
2. Anything that can be done by a DCL command should be able to be done from Fortran.
3. Anything that can be done by calling a library routine should be able to be done by using a DCL command, lexical functions or symbol.
4. Any useful subroutines lurking in the system should be available to the user.

DCL FACILITIES (lexical functions or equivalent)

Some facilities we could do with (some referred to elsewhere):-

- return user's logged-in directory
- return user's username
- return user's identification for printouts
- return user's time of logging-in
- test existence of file
- given file spec & defaults, return full file name
- given file spec & defaults, return device, directory, name, version, creation date, as separate fields.
- return user's terminal-id
- return user's CP time & connect time
- return name of last file that user referred to
- return user's disk usage and quota
- convert character string to upper case
- increment date specification by one day (or a specified time)

FORTRAN FACILITIES-

(Extensions to language, or functions)

- obtain value of a symbol (in the global or local symbol table)
- assign a value to a symbol
- add or subtract a delta-time and an absolute-time
- manipulate bits (e.g. a BIT data type, patterned after the CHARACTER data type, and notation such as variable(bit-position:bit-position))

DCL — Provide an EXECUTE command as in TOPS-20.
This is difficult to emulate effectively by a command procedure because there is no simple quick way of getting the time stamp of the files concerned in order to determine whether compilations are required - see list of desired lexical functions.

LOGIN — Don't print the System and Group login messages (see priority 0 and 0.6) if the user has seen them. This also is difficult for us to implement ourselves because of the inaccessibility of file time stamps.

DCL — Allow the default qualifiers and file-types for commands to be changed by the system manager [and preferably the user] (e.g.: SET QUALIFIER-DEFAULT PRINT/NOIDENTIFY).

DCL — Provide a means for a user to send a message to another terminal or logged-in user (TALK/USER=COM1-BLOGGS Hello there).

SUBMIT — Provide the ability to submit a command procedure daily at fixed times of the day (? SUBMIT/DAILY=(10:15,17:20)).

QUOTA — SHOW QUEUE, SHOW SYSTEM, etc, should not identify, or should show no information about, jobs belonging to users with different UICs [different groups if the user has group privilege].

QUOTA — CP time limits. In particular, the ability to have a short time limit by default on batch jobs.

VMS — Provide more than 256 groups (say 16 bits).
(This would let us allocate blocks of groups to schools, making it easier for us to delegate authorization to schools, and making group numbers less meaning less.)

FILES — Provide means of preventing the protection being changed on a directory, even by a user whose UIC matches that of the directory (see separate sheet).

VMS — Provide a facility similar to GROUPS in TOPS-20 (see separate sheet).

LOGIN — Allow periods in usernames.

AUTHORIZE — Allow accounts longer than 8 characters (say 36).

ON — The handling of ON should be controlled by the last processed ON command, and should not be altered by the occurrence of the ON condition.

SET — The effect of SET [NO]CONTROL_Y should be local to the procedure it is in, by default.

EDITOR — Provide a single editor that is

- as easy to use as the Unix editor or Univac CTS
- as conveniently powerful as the Unix editor or the University of Maryland's Univac 1100 editor.

Comment from user used to Burroughs's CANDE system:

"IBM JCL was the result of getting a command language designed by assembly-language scribblers. Clearly, SOS is the result of getting an editor designed by assembly-language scribblers."

We (Chris Bishop & Les Sullivan) feel that that remark applies even more strongly to TECO. Time for FRED (Friendly Editor).

PRINT — Print an image of the PRINT command on the flag page.

PRINT — Print a burst line at the very top and bottom of the flag page, comprising the printout identification, repeated. (Burst pages are a waste of paper, one line top and bottom is enough.)

PRINT — Require a special qualifier to allow printing of files whose type indicates that they are not intended to be printed (e.g. .OBJ, .EXE, .DIR). (e.g. if a user has files HOUSE.FOR, HOUSE.LIS, HOUSE.OBJ, HOUSE.EXE, then PRINT HOUSE.* should print only the .FOR and .LIS files.) (Our students have been wasting paper by trying to print .OBJ, .EXE, and .DIR files.)

HELP — Provide a tutorial facility. HELP is pretty useless, because it tells you what the commands do. This is back to front! Users know roughly what they want to do, they need to be able to find out what commands (and other concepts) are required to do it. Give examples and recipies, not theory.

SUSPEND — Provide a simple, foolproof way of diverting output from the normal place to a file. (ASSIGN...SYS$OUTPUT doesn't work, @TT/OUTPUT= only works at terminals, @SYS$COMMAND/OUT= doesn't work within command procedures)

ADD — Provide a simple means of taking input from a file instead of the normal place, during the running of a program. e.g.

```
! Interrupt program
! Get it to go to a file
! for further input
! EOF on MYREGRESS.DAT
! Input resumes from TTY.

$ RUN STATPROG
  Enter data:-        ! Program outputs to terminal
   1   2    3   9     ! User inputs from terminal
   ^E MYREGRESS.DAT   ! User escapes and enters input file name
  37  29    2   0     ! On EOF, input is again read from terminal
   ^Z
  192 cases read
```

MESSAGES — Make messages more informative. (E.g. in a file-not-found message, print the full file-id of the file being looked for.)

MESSAGES — Make messages less verbose. (E.g. in a file-not-found message, print 3 words: "does not exist", not 3 lines.)

REQUEST & REPLY — Make outputs from those commands less verbose and noisy.

TERMINALS — Require privilege to do SET TERMINAL/NOBROADCAST.

DEFAULTING — Extend SET DEFAULT to include a file name, type, and version, and allow this to be automatically changed where appropriate.

(Example:

```
$ SET DEFAULT  [.EXI]HOUSE.FOR
$ EDIT
Edit: DRAI:[MAPH102.BLOGGS.EXI]HOUSE.FOR;3
....
$ FOR                    ! note:
$ LINK                   !    no file specifications
$ RUN                    !    are required.
  ^Y
$ STATUS                 !
  HOUSE is running,....
$ CONTINUE
```

)

PRINT	Allow PRINT/USERID= string so that user can cause his printout to be identified by other than the normal means [which we want to be by a subfield of his username]. (This allows the user to "send" a printout to another user, simply by specifying the normal printout identification of the other user).
PRINT	Allow PRINT/LOG and PRINT/CONFIRM.
PRINT	On the flag page, display the full file identifiers of all files in the printout.
PRINT	Require a special qualifier in order to get headings printed on the first page of a file only. (PRINT/AHEADING?).
PRINT	Allow suppression of page-ejects between the various files of the printout (PRINT/NOEJECT?).
PRINT	Allow PRINT/LINE_LIMIT=n, comparable to /PAGE_LIMIT=n. (Example of usefulness: I have a Fortran program comprising 300 subroutines, each in a separate file. I want to print out the first 10 lines of each routine: PRINT/LINE_LIMIT=10/AHEADING/NOEJECT [MINITAB]*.FOR)
PRINT	By default, /NOIDENTIFY.
DCL	Allow /BEFORE=time and /SINCE=time on commands other than DELETE (e.g. COPY); and allow delta-time specifications.
FILES, DCL	Allow relative version number specifications (e.g. ;-1 to refer to the version before the highest).
PROTECTION	Provide some intermediate level between 'system' and 'user', or some privilege such as 'WORLD-READ', so that all our staff members can <u>read</u> all user files in the system, but our students in general can't.
DCL	Provide a desk-calculator facility.
FILE NAMES	Allow underscores in file names.
FILES	Provide a simple way to change the CLUSTER-SIZE on a device.
UIC	Please get rid of the UIC concept. It is not flexible enough to reflect the hierarchy that actually exists in organizations such as ours, and is difficult to work with. (Base protection on logged-in directory name, perhaps? See separate sheet.)

HELP — Provide a tutorial facility. HELP is pretty useless, because it tells you what the commands do. This is back to front! Users know roughly what they want to do, they need to be able to find out what commands (and other concepts) are required to do it. Give examples and recipies, not theory.

SUSPEND — Provide a simple, foolproof way of diverting output from the normal place to a file. (ASSIGN...SYS$OUTPUT doesn't work, @TT/OUTPUT= only works at terminals, @SYS$COMMAND/OUT= doesn't work within command procedures)

ADD — Provide a simple means of taking input from a file instead of the normal place, during the running of a program. e.g.

```
! Interrupt program
! Get it to go to a file
! for further input
! EOF on MYREGRESS.DAT
! Input resumes from TTY.

$ RUN STATPROG
  Enter data:-        ! Program outputs to terminal
   1   2    3   9     ! User inputs from terminal
   ^E MYREGRESS.DAT   ! User escapes and enters input file name
  37  29    2   0     ! On EOF, input is again read from terminal
   ^Z
  192 cases read
```

MESSAGES — Make messages more informative. (E.g. in a file-not-found message, print the full file-id of the file being looked for.)

MESSAGES — Make messages less verbose. (E.g. in a file-not-found message, print 3 words: "does not exist", not 3 lines.)

REQUEST & REPLY — Make outputs from those commands less verbose and noisy.

TERMINALS — Require privilege to do SET TERMINAL/NOBROADCAST.

DEFAULTING — Extend SET DEFAULT to include a file name, type, and version, and allow this to be automatically changed where appropriate.

(Example:

```
$ SET DEFAULT  [.EXE]HOUSE.FOR
$ EDIT
Edit: DRAI:[MAPH102.BLOGGS.EXE]HOUSE.FOR;3
....
$ FOR                    ! note:
$ LINK                   !    no file specifications
$ RUN                    !    are required.
  ^Y
$ STATUS                 !
  HOUSE is running,....
$ CONTINUE
```

)