March 1982 Vol IV No II

AUGN

Australian UNIX Users Group

N E W S L E T T E R

EXTERIOR: CRONULLA BEACH SCENE 1

An expanse of beige sandhills and blue sky.

FRONT TITLES begin and continue over the following scenes.

On the SOUNDTRACK the title song begins, "Come up to Kernel Mode".

A long stretch of beach and of surf-fringed sea. At one end apartment blocks encroach almost on to the sand. The beach here is covered fairly thickly with sunbathers and picnickers - family groups, children, men and women - and dotted with brightly coloured umbrellas. People swim between the flags. At the other end of the beach great stretches of sand dunes taper to a distant view of oil refineries. The groups of near-naked bodies are sparser here and the age groups younger. The water is dotted with

What! . . . what do you mean it's not at Cronulla, who's organising this meeting anyway . . .

EXTERIOR: HUNTER VALLEY SCENE 1

An expanse of rolling hills carpeted in lush vines.

FRONT TITLES begin and continue over the following scenes.

On the SOUNDTRACK the title song begins, "Hello Unix".

A lone rider, just a speck on the horizon, snakes his way down the long smoothly winding road. The camera switches to the distant riders view. The sound of the bike bursts onto the soundtrack, almost drowning out the title song. We switch back to our original perspective just as the rider sweeps over a low rise. The sound of the bike is suddenly just a subdued drone in the distance, the bike a speck again. An insect buzzes lazily across the cameras view and

What! . . . Katoomba! You must be joking. . . Ok, but if you change your mind once more, that's it.

EXTERIOR: KATOOMBA SCENE 1

Lots of rocks and cliffs and all that junk.

FRONT TITLES begin and continue over the following scenes.

On the SOUNDTRACK the title song begins, just use anything from the library.

The camera sweeps round all these cliffs and rocks and scree and mist and . . . Oh you know all that sort of stuff you get in the mountains. Just use your imagination. Anyway you keep this up till the titles are over. And, er, um . . . Dammit, I don't know. Write your own newsletter. . .

Yet another tiresome operating system comparison

Norman Wilson

Caltech High Energy Physics Group

ABSTRACT

A paper by David Kashtan of a few years ago presented some performance comparisons between DEC's VAX/VMS and UNIX.* These tests seemed to show a considerable advantage to VMS. This paper describes another set of tests using more recent versions of both the systems and with more emphasis towards general timesharing rather than single, huge applications. The newer tests seem to show UNIX to be generally even with VMS; in a few tests UNIX is substantially better in performance, in none is it substantially behind.

1. Introduction

This is a summary of some performance comparisons between VAX/VMS and UNIX running on a VAX.[1] They were somewhat inspired by a similar set of benchmarks run by David Kashtan of the SRI a year or two ago.[2] Kashtan was interested in the two systems' behaviour when dealing with huge images, huge data files, and lots of cooperating performances, which were expected to occur in the applications for which SRI had bought their VAX. My area of interest is more toward general timesharing, as characterised by lots of command execution (which means process creation under UNIX and often under a decently clothed VMS), and lots of accesses to probably fairly small files; i.e., the things often done during program development and text processing, which are quite different from the needs of huge applications. Therefore my tests are somewhat different, and don't exactly correspond to Kashtan's.

It should be made obvious at the outset that no decision about which operating system to use hung on these tests. I consider the programming environment of UNIX to be vastly superior to any other system I've dealt with, and certainly to be light-years ahead of VAX/VMS. I am quite willing to put up with some inefficiancy in the operating system

---

* UNIX is a Trademark of Bell Laboratories.

1. Before I forget: VAX, VMS, MASSBUS, and lots of other goodies are trademarks of Digital Equipment Corporation.

2. David L. Kashtan, "UNIX and VMS, some performance comparisons". Presumably available for the asking from SRI International, at a specific address which I've forgotten.

in exchange for the great increase in my efficiency UNIX offers. I have a very strong pro-UNIX bias; I've tried to compensate for this, but the reader should be aware of it and view my comments with suspicion.

2. Preliminary info

The tests were run on a VAX-11/780 with 4 megabytes of memory, and FP780 floating point accelerator, and an RM03 disk hooked up via a MASSBUS. The hardware was second-hand but freshly installed (which is why it was convenient to use it for benchmarks), and believed to be in good working order except for a peculiarity in the memory controller which was caught by diagnostics but seemed not to bother either operating system.

VMS tests were run under VAX/VMS version 2.2, using programs written mostly in FORTRAN. UNIX tests were run with an extensively hacked version of the system, using bits from UNIX/32V, UNIX/TS, and Berkeley's 4.0BSD. The individual programs were written in C. Some further information about how the systems were tuned can be found in appendix A.[3] For each test, two times were recorded. 'Real time' is the amount of the real-world elapsed time the test took, to the nearest second. 'CPU time' is the time spent computing, as opposed to sitting idle while the disk seeks or transfers. CPU times were measured to the nearest tenth of a second, and include both time spent in the user program and operating system overhead. Both times are expressed as mm.ss in the tables, where mm is minutes and ss is seconds.

3. Disk I/O

Almost everything done on a timesharing system in some way involves accessing a disk. Executing a command usually means reading an executable image of the system disk; text editing and program development involve many reads and writes of a user's own files. Most of this I/O usually involves going through a file sequentially (for example, a compiler reading through a program source, or the system reading a program's binary).

Kashtan's tests involved writing very large files (4000-8000 512-byte blocks), both sequentially and at random. Such things are rather uncommon in the situation in which I'm interested. Modern programming disciplines emphasize programming in small modules; this also is a convenient and useful way to organize documents. Therefore, performance in dealing with many small files is probably more important than that when using huge ones. As with any good idea, many people don't apply modularity in real life; still, there are probably very few program and document sources in existence which are as big as the multi-megabyte files Kashtan used.

---

3. The appendicies are not reproduced here but may be obtained upon request. (ed.)

3.1. I/O to sequential files

This test simply read or wrote through a file sequentially, using buffers of 512 bytes. Three sizes of files were used; 4000 512-byte blocks (read or written once), 100 blocks (with 40 passes), and 10 blocks (400 passes). Each test therefore involved 4000 blocks of data, which allowed some interesting comparisons between tests under the same system. When a file was dealt with more than once, it was completely closed and reoppened each time; thus the smaller files give some idea of the speed of file lookups as well as that of simple reads and writes.

Under VMS, sequential files with 512-byte fixed length records were used. The I/O was done through RMS, indirectly via the FORTRAN I/O library. For UNIX, 512-byte buffers were fed directly to read and write calls. Since some of the tests dealt with the same file a number of times, all of them were fed pre-existing files of the appropriate size (actually done by running the writing part of each test once before actually timing). This means that the recorded times involve only getting data out to the disk, and do not include time spent allocating blocks to a file.

	real time		cpu time	
	UNIX	VMS	UNIX	VMS
4000 blocks once:				
write	:29	:40	:27.5	:14.2
read	:43	:38	:42.5	:11.5
100 blocks 40 times:				
write	:34	:43	:33.8	:15.0
read	:31	:42	:30.2	:12.4
10 blocks 400 times:				
write	:08	1:20	:07.8	:22.3
read	:09	1:01	:08.3	:22.1

Some interesting points stand out immediately. Probably the most spectacular is the speed with which UNIX seems to be able to read small files. This is actually spurious. UNIX uses a pool of buffers as a data cache through which all file system I/O flows. The size of the cache in the UNIX kernel used for these test was 70 blocks; thus, especially with no other users around, files smaller than that will tend to stay in the cache, and the system will do almost no disk accesses after initially reading the file. Performance like this won't actually happen on a busy system, although the cache obviously helps.

A peculiarity seems to stand out in the VMS numbers as well; the amount of processor time spent on the I/O appears rather small (less than half of the real time). This is because VMS delegates a great deal of the work involved in the file system processing to a system called an 'ACP' (ancillary control process). The CPU times shown record only that of the process requesting the I/O; ACP time is not accounted for.

Less easily dismissed is the apparent strain put on VMS by dealing repeatedly with a 10 block file. Since it seems unlikely that there's a special problem involved in small files, the next best guess is that

looking up a file is quite expensive.

I/O in UNIX centers round a structure called an 'inode', which contains such information as file permissions and modification dates, and also the block numbers for the first ten blocks of the file. The remainder of the file is mapped out in 'indirect blocks'; the eleventh pointer in the inode points to a block full of pointers to blocks full of pointers, and the thirteenth to a block of blocks of blocks of blocks.

The inode remains in-core for each file that some process has open. This explains why UNIX reads 4000 block files a bit more slowly than 1000 block ones; the additional time is spent fetching blocks full of pointers.

VMS, on the other hand, uses a structure called a 'file header', which contains its retrieval pointers directly. Parts of a file which happen to be contiguous on the disk can be described with a single pointer, unlike UNIX where each block of the file is described independently. Unless a file is extremely large or is very fragmented, the file header fits into a single disk block. Thus VMS doesn't have much additional work in deciding to which blocks belong to a 4000 block file, and seems to be slowed down a little rather than sped up by the 100 block file.

None of this explains why UNIX was able to write 4000 blocks as quickly as it did; this is not yet understood.

Under VMS, it's possible to set a per-process number called the 'multi-block count' to cause more than one block to be read from the disk at a time. The numbers above were run with the default value (which is rather small; 4 to be exact), under the assumption the DEC had some reason for setting it as they did.[4] For comparison, however, the VMS tests were run again with the multi-block count set to 32 (the number of sectors per track on the RM03 disk that was used). This yielded the following numbers:

	real time		cpu time	
	VMS(4)	VMS(32)	VMS(4)	VMS(32)
4000 blocks once:				
write	:40	:11	:14.2	:09.6
read	:38	:10	:11.5	:07.0
100 blocks 40 times:				
write	:43	:17	:15.0	:10.8
read	:42	:13	:12.4	:07.9
10 blocks 400 times:				
write	1:20	1:00	:22.3	:18.4
read	1:01	:34	:22.1	:17.5

With large, and even moderate-size files, the performance is very

4. Not to mention the likelihood that few users will think of changing it!

impressive: VMS with a multi-block count of 32 reads a 4000-block file in essentially the time UNIX can copy a 10-block file 400 times from its in-core buffers. However, for small files, the performance is not amazingly good for reads, and still quite dismal for writes. Evidently something other than reading the disk is dominating the time spent.

3.2. Random-access I/O

I don't really feel that random access, especially to large files, is terribly important for timesharing. However, as it was convenient and would be interesting to compare with Kashtan's figures, a test was run consisting of picking a block a random from a 4000 block file and writing it. VMS relative files were used.

	real time		cpu time	
	UNIX	VMS	UNIX	VMS
10000 random accesses:				
write	4:02	3:12	3:58.5	:40.7

VMS appears to have a bit of an edge, presumably because it needn't do disk accesses just to find the block number it wants. The amount of work done by ACP under VMS is again evident.

3.3. File lookups

The simplest explanation that springs to mind for the slowness of VMS when dealing with small files is that searching for a file and opening it is very expensive compared to UNIX. It's reasonable that this might be so, as the two systems are very different in the way they handle the concept of the 'working directory' (the directory with which the user is currently dealing, from which files with no explicit directory name are read). UNIX keeps the inode (described above) for each process's working directory in memory, and keeps track of that directory by simply recording the address where the inode is stored. Thus, looking up a file in the working directory means searching just that directory, without worrying about its parents. VMS, however, stores an ASCII string giving the name of the current directory; each time a file in the working directory is referenced, the saved string is prepended to its name, so that all directories and subdirectories in that path are searched. For example, the file looked up in the test below was called [norman.bm]foo; to find this file, VMS must first search [0,0] (the master directory for the volume) for norman.dir, searches that for bm.dir, and finally looks for foo. Recent versions of VMS have caching schemes for such things a blocks of directories, which are meant to speed up this process.

To try to understand how VMS is slowed down by extra directory searches, UNIX was asked to search for the file a/b/c/d/e/f/g/h which has eight directories named. Since the directory names are explicitly specified, UNIX is actually obliged to go out and look at each of them. VMS was fed an additional file as well; it was asked to look up [0,0]indexf.sys, which is a file in the master directory of the disk volume and therefore requires only one directory search.

Lookups under UNIX were done by invoking the open call; under VMS, the FORTRAN open statement was used. In both cases, the programs claimed that they wished only to read the file (mode 0 was used under UNIX, readonly was specified under VMS).

	real time		cpu time	
	UNIX	VMS	UNIX	VMS
10000 file lookups:				
current directory	:56	6:03	:56.4	3:42.9
a/b/c/d/e/f/g/h	5:21		5:20.6	
[0,0].indexf.sys		5:44		3:38.7

Even when its best case is compared to UNIX's worst, VMS looks pretty bad. It could be claimed that UNIX's buffering scheme makes it faster, but VMS is meant to be caching too when dealing with directories. Some time could be being wasted by some obscure part of the FORTRAN runtime library, but this seems unlikely when the real times are compared to the CPU times. Either much time is being spent waiting for the disk, or a lot of work has been done by the ACP for each lookup.

3.4. Text files

The I/O test described above were done with big buffers and (under VMS) huge records in the file. This is somewhat unrealistic. Although both systems access files a block at a time when executing programs, much I/O under a timesharing system will be done to files containing more structured data like ASCII text. Under VMS, files with small, variable-length records are generally used for this sort of data. UNIX doesn't impose a record structure on files, but most programs do their I/O through a standard library on a character-by-character basis; the library actually does reads and writes of 512-byte buffers, but additional overhead is involved in handing out individual bytes.

Since VMS tends to think in records while UNIX works with characters, it's not clear how to compare the two. It's not even clear that it is reasonable to do so, since programs under the two systems tend to be written somewhat differently because of the different I/O approaches. A reasonable guess seemed to be to read a file as records under VMS, and as both records and characters under UNIX. accumulating a line at a time. In real life, UNIX programs don't do this very often, but it seemed fairest to include it.

The file dealt consisted of the same line of 35 characters of text (plus newline) repeated 30000 times. It was written a line at a time under both systems. (Again, UNIX actually does things a character at a time, but for output it's quite common to feed a lineful of text to a formatting routine, which then send out the bytes). In both cases, the file was created afresh rather than just being overwritten as in the big block tests. The file was then read in a record at a time under VMS, and both byte-by-byte and line at a time (via getchar and fgets respectively) under UNIX.

	real time		cpu time	
	UNIX	VMS	UNIX	VMS
30000 text lines:				
write (lines)	:31	:24	:30.7	:21.8
read (lines)	:26	:27	:26.1	:26.7
read (chars)	:25		:25.6	

The two systems are about even for input, but UNIX is a bit behind for output. It seems rather strange that UNIX writes faster than it reads when dealing with big blocks, but reads faster when dealing with text. Perhaps fprintf, which was used to format the output, is slow; or perhaps UNIX is slow at allocating blocks to files.

This test was also run under VMS with an RMS multi-block count of 32. The results were hardly different:

	real time		cpu time	
	VMS(4)	VMS(32)	VMS(4)	VMS(32)
30000 text lines:				
write (lines)	:24	:22	:21.8	:21.5
read (lines)	:27	:26	:26.7	:24.8

4. Simple interprocess communication

One of the more unusual features of UNIX is the ability to connect multiple processes in such a way that the output from one becomes the input of another. Although program development and text processing don't greatly need a more general IPC, this 'pipe' mechanism turns out to be extremely powerful, and is frequently used. To be worthwhile, pipes require reasonably fast process creation, and a reasonably efficient interprocess communication facility.

Although the designers of VMS don't seem to have thought of pipes a such, there are corresponding (although messier to use) mechanisms. They become quite important in UNIX-emulating packages such as INTERACTIVE's IS/1 and SRI's EUNICE. The closest equivalent of the pipe under VMS is the 'mailbox'; this was used for the tests below. It is also possible to construct one's own IPC facility using shared memory under VMS; Kashtan did this as he was dissatisfied with the performance of mailboxes, but it seemed too much trouble to bother with here.

To get the following numbers, 10000 512-byte blocks were shoved through a pipe on each system. First a process was asked to write into a pipe and immediately read the result back; later, writes to a pipe being continuously read by another process were tried. In the latter case, only the writer was timed.

	real time		cpu time	
	UNIX	VMS	UNIX	VMS
10000 blocks through a pipe:				
write to self	:23	:29	:23.7	:23.6
write to other	:27	:36	:13.8	:14.0

UNIX seems to be a bit faster. Strangely, VMS uses about the same CPU time as UNIX, but more real time. This can't be explained the same way as for disk I/O, since mailboxes don't have ACPs.

UNIX will buffer up to 8 blocks of data in a pipe, VMS has the amount of buffering specified when the mailbox is created. Since only 512 bytes were asked for in the numbers above, it was decided later to re-run the VMS tests with 8 blocks of buffering. The results were:

	real time	cpu time
10000 blocks through a pipe:		
write to another process, 8 block buffers	:34	:13.0

The difference seems insignificant.

5. Command execution

Execution of user and system programs is obviously something that happens frequently in a timesharing environment. It's difficult to compare UNIX and VMS directly in this, as the way commands are executed differs a great deal between the two systems. In UNIX, a new process is created every time a program is run; under VMS, a magical command interpreter usually invokes different images within the same process. Process creation is still important under VMS if a UNIX environment is being simulated, although in practice it's often unnecessary.

With this in mind, both systems were asked to create a new process executing a program which immediately exited. This means a fork and exec for UNIX, or a SYS$CREPRC under VMS. VMS was also asked to execute the same image a lage number of times within the same process. This was controlled by a small program written in the language of the VMS command interpreter, DCL. Since DCL is notorious among its users for being slow to execute some kinds of control flow statements, the same command file was run again without the image invocation to get an idea of the overhead involved.

Under both systems, the do-nothing programs were written in assembler to avoid any overhead from language-specific runtime startup code. On VMS, the process creation program was written in assembly language as well, since the system calls involved are extremely unpleasant to use from higher-level languages.[5]

5. Actually, they're no treat in MACRO, either.

On both process-creation tests, the CPU time shown is that of the creator. The real time accounts for everything, though, as the parent always waited for the child to die before creating another.

	real time		cpu time	
	UNIX	VMS	UNIX	VMS
10000 processes:				
	6:27	42:46	6:21.9	1:37.0
10000 image activations				
benchmark time		19:26		13:22.9
DCL overhead		1:44		1:44.4
difference		17:42		11:48.5

The immediately obvious thing is that UNIX is very much faster at creating new processes. As with the disk I/O, VMS seems to show very little of the CPU time involved; this is probably related to the fact that the job controller (a separate system process) is doing some of the work, but this is not well understood.[6] The most surprising result is that VMS is almost a factor of three slower in executing a new image in an existing process than UNIX in creating a new one. This isn't well understood either.

6. Tests that were not run

Kashtan compared paging performance of the two systems, when dealing with various types of faulting behaviour in a four magabyte process (on a machine with two megabytes of physical memory). I consider this to be of little interest for a general timesharing load, as I've observed on my own system (running UNIX, with four megabytes) that memory is almost never seriously short; in fact one or two megabytes are usually sitting around free![7] Memory is cheap enough these days that high-performance paging seems irrelevant, at least with moderate timesharing loads on a VAX.

Kashtan also did tests of context-switching performance, and of various flavours of IPC. These are of interest for real-time applications, but not for normal timesharing loads, unless performance is exceptionally bad.

7. Tests that should have been run

The overhead associated with terminal I/O, particularly fairly high speed output, is reasonably important for timesharing use. However it's difficult to think of a way to get good numbers for this, at least without poking hooks into the operating system. Further, the only terminal interface on the VAX available for testing (other than the console) was an ABLE DH/DM; UNIX can use it, but no driver is available for VMS.

---

6. In fact, I don't well understand how VMS goes about creating a process.
7. SMP notwithstanding.

All the tests described herein were run on a single-user system. They would be much more useful if run under some kind of simulation of a multi-user environment. Since this is non-trivially hard, and there were significant time constraints, it was decided not to bother with this.

These tests were designed only to try out some of the primitives implemented by the operating system kernel. The overall performance and desirability of a system are determined by many other things, such as the ease of the use and the quality of the available utilities, especially compilers.

8. Non-conclusions

Given my acknowledged bias toward UNIX, I'm wary of trying to draw conclusions; I'd rather leave that as an exercise for the reader. I don't think any of the data shows VMS to be terribly exciting, though. Certainly none of the results make me want to switch operating systems, but this is hardly surprising.

A bastardized paging UNIX*

Norman Wilson

Caltech High Energy Physics Group

ABSTRACT

One of the VAXes in the Caltech High Energy Physics Group runs a peculiar hybrid of several versions of UNIX. Its principle claim to fame is that it is recognizably a variant of UNIX/TS, but has virtual memory code grafted in from Berkeley's 4BSD. It is also compatible with the UNIX/TS system run on our PDP-11/45.

This paper is meant to give some idea of what our system is, why it exists, and how much effort went into its creation.

1. Intruduction

Our system, which for barely good reasons is referred to as UNIX/VM, started life as UNIX/TS. This is a Bell Systems internal release of UNIX which Caltech had obtained under special agreement, as part of a joint project with the local Computer Science department.[1] TS is a somewhat cleaned-up descendant of Seventh Edition UNIX; despite a few dubious features, it is small, simple, reliable, and fairly convenient to use and administer. Particularly important to us are a very simple and straightforward mechanism for specifying the hardware configuration to the kernel, and a high degree of compatibility between the systems on the VAX and the PDP-11; for example, we have many device drivers which run essentially unmodified[2] on both systems.

The problem with TS is that its VAX incarnation is a swapping system; to be runnable, a process must be fully resident in physical memory. For much of our load this is fairly irrelevant, but a few of our users need to run with huge address spaces (particularly a group working on a large symbolic manipulation program). It therefore became inevitable that we run a system with virtual memory support. After considering various alternatives, we decided to put paging into our TS system by lifting the virtual memory code out of Berkeley's Fourth Software Distribution (4BSD).

---

* UNIX is a Trademark of Bell Laboratories.

1. UNIX/TS has now been released as UNIX 3.0 (ed.)

2. Between zero and five lines of difference; the changes involve setting up UNIBUS mapping registers on the VAX.

The paging conversion accounts for the greater part of the difference between our system and normal UNIX/TS. Other changes include performance improvements from various sources, a few bug fixes, and some miscellaneous fiddling.

2. Paging

2.1. Why bother?

On first thought, it seems wasteful and silly (not to say un-UNIX-like!) to create one's own paging kernel when one already exists. Some elaboration of why we did so anyway is probably in order.

Paging first became apparently (though not urgently) necessary in early 1980. At that time, there were two versions of UNIX with virtual memory support in existence for the VAX. John Reiser of Bell Labs had a paging system, but it was claimed to be significantly unclean as yet and in any case was very difficult to get since it was not officially licensable from Bell. The University of California at Berkeley had an easily available paging system. However, early versions of this system (3BSD) did not sound very worthwhile: different sites running it gave us widely differing reports of its reliability, performance, and for that matter whether it would run at all! As we had other things to think about anyway, the paging issue was sidelined for a while.

In fall[3] of 1980, Berkeley came out with a new release (4BSD). This system was much more reliable and had quite good performance. Unfortunately, Berkeley had done a great many other things to their system; the result was both unpalatable and impractical to us.

Our objections to straight 4BSD can best be divided into the following areas:

Portability - As mentioned above, we have a PDP-11 as well as a VAX. We not infrequently move devices between the two machines. It is also extremely useful to have essentially the same system running on all computers; it avoids confusion for users who might otherwise have to learn two ways to do things, and makes for a lot less work for system people who have only one set of system software to worry about rather than two. Thus, the high degree of compatibility between the VAX and the PDP-11 flavours of TS is of great importance to us.

Berkeley, however, seem somehow set on making UNIX as non-portable as possible. Their VAX kernel is full of explicit VAX dependencies, such as overuse of the asm keyword and the way they've rewritten the UNIBUS and MASSBUS interfaces, and implicit ones such as the huge amount of code added for features of doubtful utility.

---

3. Their Autumn - roughly our spring (ed.)

Stability - Berkeley seem set on changing many fundamental parts of the system; in the past they have redesigned a.out format, rewritten the terminal driver and the shell, and twice made it necessary to dump and restore all filesystems. They warn that further such changes (in particular, yet another filesystem rewrite) are iminient. Our users are not interested in relearning the system with every release, and our system people aren't fond of the idea of dumping and restoring 800 megabytes of disk every few months.

To be perfectly honest, at the time 4BSD came out we had been running TS for some time. We therefore weren't too fond of even one drastic changeover unless it promised great benefits.

Philosophy - We feel that many of Berkeley's changes to the system and the supporting programs are unnecessary, un-UNIX-like, and just plain silly. To take a favourite example: anyone who will add a flag to cat to make it number lines, remove blank lines, and make control characters visible, rather than writing separate filters, simply doesn't understand what UNIX is all about.

We needed paging; it would have been silly to attempt to write it ourselves; but the only available paging system was one we didn't want to use. It therefore seemed correct to pick out those parts of the paging system we wanted, and insert them into the system we liked.

2.2. Chronology

To give you some idea of the amount of time and effort involved in the conversion, here is a rough chronology of the paging effort. It's not especially accurate, given that this document is being written about a year after the fact.

The real effort towards a paging system started in Feburary of 1981. Initially, I wasted some time in an attempt to insert the desirable parts of TS into 4BSD; this was abandoned when it became apparent that the majority of TS was desirable. This effort was, however, useful in that it taught me a great deal about the innards of 4BSD.

After a month or so of work without much tangible progress, I decided to back off and consider the opposite approach: rather than inserting the nice parts of TS into 4BSD, graft 4BSD paging code and some of its performance improvements onto TS. I spent two weeks comparing the systems module by module (and doing other things to get my mind off the problem!). This comparison convinced me that this was the right approach; the real work was undertaken, and within a few weeks more I had a bootable system.[4] It took about a month of exercising the new system to shake the more obvious problems out of it. The paging system came up for users sometime in June, and has run fairly happily ever since.

4. Although init dumped core the first time I brought it up.

3. Other local changes

Every UNIX site hacks their system a bit; we've probably done so more than most. As a result, UNIX/VM contains many local changes, some of dubious utility. Except as noted all this code was conceived and written locally; although parts of it duplicate functions in other systems (eg 4BSD also knows how to autoreload), our code was invented independently and generally before hearing of other people's attempts at the same goal.

3.1. Autoreload

Normal versions of UNIX loop forever (until someone comes to help) if the system detects an inconsistency and calls panic. In our system, panic has been hacked to request an auto-restart and then halt (allowing restart to occur). Recursive calls to panic cause a halt without restart. Machine checks and invalid kernel stack traps also halt without restarting; the assumption is that these are caused by errors serious enough that someone knowledgable should check things out before the system is brought back.

The autoreload just causes the console to execute restar.cmd off its floppy; currently this loads a standalone program which copies all of memory to a known place on one of the disks, then reboots.

3.2. Push-button reload

In conjunction with the auto-restart, the system has been fiddled so that if someone just types BOOT to the console, the system will come up to multi users mode (after checking filesystems for consistency) wihtout user intervention. This involves perhaps two lines in the assembly language start routine and three in icode, all involved in passing info to /etc/init which does the real work.

3.3. Terminal driver

Everyone rewrites the terminal driver. So we did. Among the sillier things are several more control characters and infinite settability; among the useful ones, a few fiddles to make the system know the difference between a hardwired line run with three wires (for which the carrier is always true), a line connected to an answer modem, and one connected to an originate modem. Code for this sneaks into tty.c because it really is device-independent, and we are using two different kinds of interface.

I have regrets about what I did to the terminal driver when I was young and foolish,[5] and intend to clean it up someday . . .

5. Well, more so than I am now, anyway.

3.4. Configuration cleanup

Some VAX-dependent parts of device configuration and initialization have been cleaned up. Our code allows the TS-style configuration file to specify which MBA a MASSBUS device will live on, instead of building it into the driver and an assembly language interupt dispatcher. When the system boots, it is reasonably careful about making sure I/O adapters really exist, and prints a specific message (and still comes up) when they don't instead of bombing with a machine check. These seem trivial changes but have come in handy on a few occasions.

3.5. Bug fixes

A few bugs existed in the TS system as it arrived. The worst of these were a problem in the mknod system call which sometimes left processes hung instead of returning EPERM, and two lines of code which had to be interchanged to prevent pipes from garbling data when the sytem was busy. It's worth noting that both of these bugs were in code new to the release of TS we had; someone inside Bell Labs needs to learn care in adding features, too.

4. Ideas which were thrown out

It might be interesting to touch briefly on some parts of Berkeley's system that weren't adopted.

4.1. Autoconfiguring

This seems a lot of work and bother for a trivial result. TS is connfigured quite cleanly at system build time (all device and vector addresses live in one file); we consider this quiet sufficient for our needs. The only time it matters is when a device disappears for a while, and all our current drivers are sufficiently robust to handle that as long as noone actually tries to do I/O to a broken device.

4.2. Filesystem blocksize

Berkeley have changed the filesystem to use 1024-byte blocks, and claim that this is the only possible way to make their system work, We have found the latter to be untrue, and in any case consider the benefit of huge blocks to be offset by the conversion time, the lack of compatibility with other UNIX sites (including our own 11/45), and the many insidious effects of such a change unless Berkeley's entire system, including their utility programs, are adopted (eg, a.out format depends on blocksize). Disk I/O is certainly the traditional bottleneck for UNIX systems, but we feel there are more general and useful ways to attack the problem, such as enlarging the buffer cache.

4.3. Vread and vwrite

These calls are of limited apparent utility, poke dirty fingers all over the code that was formerly clean, and claimed by the Berkeley people to be deprecated; we therefore avoided the Christmas rush and removed them now.

4.4. Etc

Job control, complete rewrites of the UNIBUS device driver interface, a second signalling mechanism, two or three terminal drivers at once, polling DZ11's for input in the clock interrupt routine, teaching the device drivers the ASCII names of all the error bits, spewing the configuration on the console whenever you reboot, are symptoms of a philosophy which we do not share. Systems should be small, simple, and correct; features such as the above aid none of these goals.

More on 11/70 UNIX* profiling+

Kevin Hill
(kev:elecvax)

Bob, some profiling results I have just obtained from the 11/70 system that may be of interest to AUUGN readers. The profile was taken during our last week of session, with the machine fairly loaded, for a period of almost 30 minutes. Only those routines scoring over 2% of the kernel time are shown, sorted by %. It should be emphasised that the profiling is only approximate due to its coarseness of one longword count per 4 bytes of text (e.g. note the 0 calls recorded for sched)[1].

Name	#ncall	%time
cret	6016	8.3
_dzstart	1078	5.3
_dzxint	608	5.2
call	1556	4.9
csv	5810	3.8
_getc	1330	3.6
_putc	1207	2.9
_trap	171	2.7
_ttyoutp	750	2.6
_sched	0	2.5
_splx	4902	2.4
_swtch	106	2.4
_fuword	466	2.3
_dzscan	38	2.2
_sureg	99	2.2

Plainly, there are no real hogs of CPU time as such. Some minor work can be done to speed a few routines up, but will not result in vast improvement.

---

* UNIX is a Trademark of Bell Laboratories.

+ See "A brief note on UNIX system preformance", AUUGN Volume 1 number 4, May 1979.

1. "ncalls" is guessed from the number of clock ticks accumulated at function entry. (ed.)

The % times spent at various priority levels were as follows:

Level	%
0	43.7
1	0.4
2	0.0
3	0.0
4	1.6
5	33.4
6	20.9
7	unmeasurable

The amount of time spent at level 6 is of interest - this includes clock interrupts as well as any spl7( ) calls from the C code (most of which were altered to set the priority to level 6). Obviously the amount of time at level 7 cannot be determined, as the profiling clock interrupted at this level.

Of final interest is the proportion of time spent in kernel mode to the amount spent in user mode - this was 78% to 22%. This ratio was much lower under our old level 6 system - something like 60% to 40%.

More, and different, profiling is planned for the coming year.

Informal UNIX machines survey

If you are in some sense responsible for a machine running UNIX, or if you feel that no-one else will bother, we would like you to answer these questions. We're not going to give you the chance to object to having the results published, just don't say anything you don't want known. Send the completed questionnairs to:

AUUGN
c/o Bob Kummerfeld
Basser Department of Computer Science
Madsen Building F09
University of Sydney
NSW 2006
Australia

Photocopies are fine. If you're on the network you can netsend the results to auugn:basservax. Blank forms can be found in %auugn/survey on both basservax and basser40.

Who are you? (You, your institution and your relationship to it.)

Why do you have a computer anyway?

What is it used for? Who and how many are your users?

What type of licence do you have, and what versions of UNIX are you licenced to run?

What version of UNIX do you run? (Give us some history if you can, and an idea of how much you've hacked it.)

Tell us all about your configuration. Things like:

a) Your CPU (give a cpu number in case of multiple returns), its memory and attachments of note. . .

b) Total mass storage, and the hardware involved. . .

c) Tape facilities, hardware and how you like your data stored and transfered. . .

d) What sort of terminals and how many do you run. Don't forget the baud rates. . .

e) Printers?

f) Graphics?

g) Network hookups and miscellaneous links?

h) Anything else you'd like to say. . .

Proposed extensions to the VAX instruction set

Karlos Mauvtaque

There really are lots of goodies for the assembly language programmer tucked away in the VAX instruction set. However there are a number of areas where a little more life could be injected.

Character String Instructions

Yes we have MOVTUC, and CMPC3, and those ridiculous LOCC and SKPC. To help DEC New Products marketing division to document their hardware, their microcode hackers really should be working on MOVTACS - Move Translated and Correct Spelling. Truly a must for the para-serious text processor.

Special Instructions

I was really impressed by the wide range of bit field instructions, and REMQTI (pronounced rem-cutie) really knocked me out! But for the serious system programmer there is definitely a need for multiple register processor status self relative variable length bit address manipulation instructions. I propose POPEXTPSLATI, that's Pop and Extract Field from Processor Status Longword Address At Tail Interlocked.

For those difficult conversions you need CVTPUF, Convert to Previously Unthought-of Format which can really benefit Cobol compilers. Note that a "That's not been unthought of" fault can be generated from this instruction.

The FFB, find first bug, is an essential addition to the 'go get me one of these' instructions. And for those difficult flow control constructs why not ACBFC, Add Compare Byte and Forget Context, surely an instruction for the Pascal 'goto 99' connoisseurs.

Procedure Calling

In addition to the infamous CALLS and CALLG we need a CALLV (Call with Virtual argument list) for Pascal programmers who like to pass around global variables as var parameters.

Information hiding, or 'eyes off my private parts[1]' will play an

1 See iAPX 432 Object Primer

important role in the future of computing. Thus a RIPE instruction, Return Ignoring Previous Error, is necessary to hide those private little errors that you don't want the rest of the program to know about.

Addressing Modes

Well I don't have a lot to add here. Nine out of ten Californian Programmers can't tell the difference between autoincrement deferred indexed mode and a dead rat. I would however quite like an Absurdly Deferred Addressing Mode, O^#*@(Rn!), for those difficult indirections.

System Instructions

For the Software Reliability guys out there, you need TSTPF, Test Probability of Fail-because-I'm-a-750. This is quite essential for those MOVTUCs with overlapped operands.

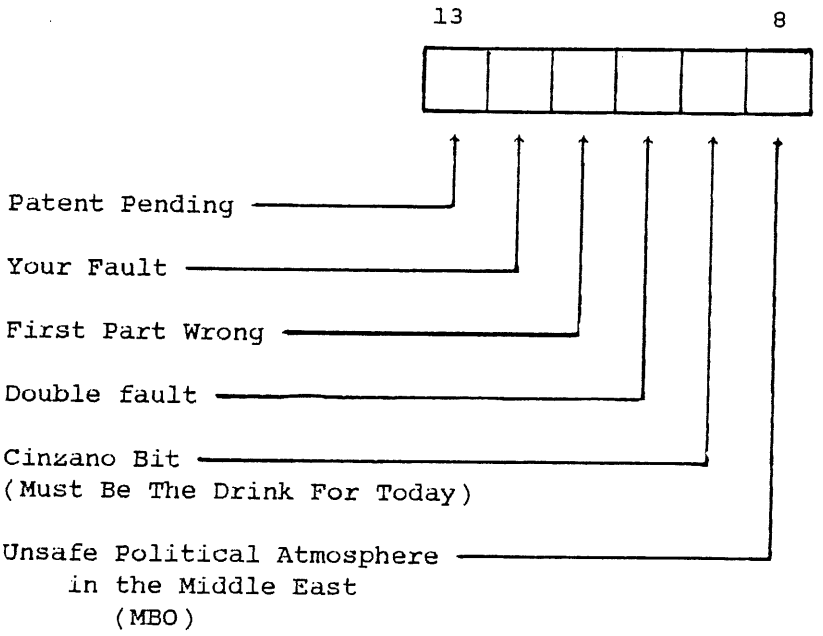
For Richard Grevis and friends we need LDPCAV, Load PC with Absurd Value. You can then make your VAX croak without having to resort to the LSI-11. Others in this field include ROCPUB - Rip Out CPU Boards, TOAC - Turn Of Air Conditioning, oh I don't know, use your imagination.

The CTPR, Convert To Privileged Register, would encourage the casual user to come on up to Kernel Mode to access those intimate variables. CSBR, Complement System Base Register, is a must for those esoteric resource allocation schemes.

The critics of the VAX memory management would go bananas over a PMPP, Predict Most Popular Page, instruction. A whole family of psychic resource management instructions could be implemented to make life a lot easy for that poor tired operating system.

Processor Status Longword

Those well worshipped MBZ bits in the PSL can be replaced with new and exciting status information.



Hardware Upgrade

The hardware could be spruced up a bit. There is a great need for a VAX 11/750ml CPU (that's Capuccino Processing Unit) so we can upgrade from our current heavily overworked model. A PT11 Precognitive Terminal Interface would help handle the student load and complement magnificently the memory management instructions.

Data Types

I feel that the current move towards exotic data types like the recently announced 'octaword' is generally misguided. A pentaword specifically designed for the United States Defence Department would be a welcome addition.

Emulation

One of the sweetest gestures by DEC designers was the PDP11 emulation. I propose emulation of the Turing Machine, the Analytical Engine, the IAPX 432, and the Wankel Rotary Engine. This would solve a real need.

UNIX ● XENIX ● ONIX ● IS/1 ● SYSTEM III ● MEMIX ● VENIX ● UTS ● TNIX ● IDRIS ● CROMIX ● UNIFLEX ●

● OS/9 ● COHERENT ● OMNIX ● DYNIX ● GEMINIX ● OS-1 ● UNOS ● UNIX ● XENIX ● ZEUS ● ONIX ●

Do you use UNIX*

or any Unix-based or Unix-like system?

You may be interested in the

European Unix User Group

Our next meeting will be held in Paris where we shall also be holding an Open Meeting. At this the major vendors of Unix and Unix-like systems will be exhibiting and giving presentations on their products. For further information on membership of the EUUG contact

Hugh Conner
EUUG Membership Secretary
Department of Electrical and Electronic Engineering
Heriot-Watt University
Mountbatten Building
31-35 Grassmarket
Edinburgh EH1 2HT
Scotland

and for details of the meetings

Open Meeting — April 14th
Ian Perry
Group Informatique
L.E.R.S.
58 Rue de la Glaciere
75013 Paris
France

Members Meeting — April 15th/16th
Bernard Martin
Laboratoire d'Informatique
CNAM
292 Rue St. Martin
75141 Paris
France

*UNIX is a trademark of Bell Laboratories

● COHERENT ● OMNIX ● DYNIX ● GEMINIX ● OS-1 ● UNOS ● UNIX ● XENIX ● ZEUS ● ONIX ● IS/1 ●

● OS-1 ● UNOS ● UNIX ● SYSTEM III ● MEMIX ● VENIX ● UTS ● TNIX ● IDRIS ● CROMIX ● UNIFLEX ● UNIX ●

Existing Unix Users should contact their installation correspondent for further details on either of the above meetings.

1982 Application for New Membership
USENIX Association
Individual or Public Membership

[ ] Individual ($12, Institution has Source License)
Institutional Affiliation: _____
Nature of Affiliation: _____

[ ] Individual ($12, Institution has Binary License Only)
Institutional Affiliation: _____
Nature of Affiliation: _____

[ ] Public ($12, Not covered by Non-Disclosure)

Mailing Address (Individual Members must use institution address):
Name: _____
_____
_____
_____
_____
Phone: _____

[ ] Overseas airmail, add $5.00

[ ] Invoice required, add $3.00 bookkeeping charge for invoice or receipt
[ ] Receipt required

Check enclosed: $ _____

Return completed form to:

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

(For Institutional Membership or membership renewal contact the Association offices at the address above or at 212-570-8934.)

DEAR ABBY DEAR ABBY DEAR ABBY DEAR ABBY DEAR ABBY DEAR ABBY DEAR ABBY DEAR ABBY

From gregm Mon Mar 8 18:25:53 1982
To: abby
Subject: Dear abby

I don't understand why the VAX doesn't know the time (all the time). If Nifty (*) can make sense of it why can't Vaccy?

(signed) Worried.

Dear Greg,

Basically it all comes down to the fact that 1 January 1970 was a Thursday. (You have to know these things when you're a systems programmer). The trouble starts when programs try to be internationalist by interpreting the UNIX-wide date (number of seconds since 1 January 1970 GMT) themselves, instead of using a standard library to interpret it for them. This is why it was never considered prudent to nroff a document near four o'clock in the afternoon, because your document may get a date revision between pages six and seven. We have now fixed this bug, despite the cries of outrage we receive when old-timers come barging into the system supervisor's office at midnight complaining that nroff thinks the date has changed. We hope that we have weeded out all the programs which look at GMT (over which Nifty Nev has no juristiction).

Yours very truly,
Abby

(*) "Nifty" is the nickname of Neville Wran, the Premier of New South Wales, renown as a champion of power-saving extentions to summer time. He has resisted the many arguments against it, which have won in the darker state of Queensland. The more interesting of these arguments assert that it fades curtains, makes grass grow excessively, as well as disturbing the daily rhythms of cows and computer systems.

From 8153758 Thu Mar 11 08:26:14 1982
To: abby
Subject: UNIX

Dear Abby,
Is it true that UNIX with paging (virtual memory) would be no great advantage as what was gained in memory space would be lost in speed?

Yes, Virginia, there is a trade-off. If Sydney Uni ran a popular paging system instead of our local system, the maximum number of users would drop from 85 to about 40. Of course, those lucky few who could log on would be able to run programs much larger than 200k, and root might even be able to run objects larger than 1M! At the moment we have to give up such dreams so that our large student population can actually use a computer.

Yours very truly, Abby

From doug Mon Dec 21 13:40:08 1981
To: abby
Subject: your letter

Your reply was a great help to me. It bucked me up and put new feeling into my tendonitis-ridden typing finger. I once more thrill to the sight of my latest achievment: 'hello world'.

Many Heartfelt Thanks,
Doug

From root Fri Mar 19 11:23:10 1982 netmail from chemeng
To: abby:basservax
Subject: Compatibility

How can I explain to the users of my system the lack of consistency introduced by renegade software from UCB?

Why would anyone write a program that operated exactly opposite to something it was trying to replace?

I refer of course to the command 'nice', now intrinsic to Bill Joy's cshell.

The manual entry that I have for nice describes its usage as

nice -number command [arguments]

but what does Mr Joy's shell do?

Well when you say 'nice -10 fred' to the cshell it generally ignores your intention of lowering the priority of your job (it won't of coure tell you of your 'mistake') unless you happen to be the super-user in which case it embarrasingly paralyses the system.

How do I break the news to my users that you really must type 'nice +10 fred'? How do I explain this?

What is next on the agenda from UCB? Perhaps changing 'rm' so that it removes all of the files not named or changing the ordering of the arguments to 'cp'?

Deus.

Dear Deus,

I receive many troubled letters about the Californian software and entertainment industries, mainly from people who have followed their practices and find they can't handle it. The obvious solution is not to use Mr Joy's shell. If you are addicted to its powerful features, you must either learn to cope with its side effects, or find a substitute. I notice that people at Sydney Uni are working on a new shell, perhaps you could mail your complaints to chrisb:basservax.

One can hardly expect the UNIX philosophy to survive unchanged the voyage across the American Continent, let alone the Pacific Ocean. The people, their needs and their ambitions are different. The best we can do is look at each other's work, and use what seems best for local conditions. This may entail writing local software. The moral here is to document it well. Don't worry too much about UCB, they can take care of themselves. As for your users, you could try telling them that just as the seasons are reversed in the Northern Hemishpere, so the nice priorities, too are reversed. I don't think this is very meaningful, but it will probably help them remember it, or at least confuse them sufficiently that they won't bother you.

Yours very truly,
Abby

From peteri Fri Mar 19 09:57:35 1982 netmail from elecvax
To: auugn:basservax
Subject: do you have a copy of this?

From pwbcc!andrew Tue Mar 16 16:07:37 1982 netmail from usa

COMP SCI SERENADE
SUNG TO THE TUNE OF MY BONNIE LIES OVER THE OCEAN

1) MY PROGRAM LIES UNDER THE BACKLOG,
MY CARD DECK'S ALL OVER THE FLOOR,
THE PLOTTER IS USING A CRAYON,
AND I JUST CAN'T TAKE ANYMORE!

CHORUS: BRING OUT, BRING OUT,
OH, BRING OUT MY PRINTOUT TODAY, TODAY!
BRING OUT, BRING OUT,
THE ONE YOU RIPPED OFF YESTERDAY!

2) THE CARD READER CHEWED UP MY JOB CARD,
AND SOMEONE ERASED ALL MY FILES,
THE SYSTEM HAS BEEN DOWN FOR HOURS,
WHILE PEOPLE COLLAPSE IN THE AISLES.

CHORUS: FLUNK OUT, FLUNK OUT,
I WORKED LIKE A DOG EACH AND EVERY DAY.
FLUNK OUT, FLUNK OUT,
TWELVE PROJECTS WERE DUE YESTERDAY!

3) SECURITY HOLES I'VE DISCOVERED,
THE RECORDS OF GRADES ARE NOW MINE,
WHAT ONCE WAS A ONE POINT FIVE AVERAGE,
WILL SOON BE A THREE POINT NINE NINE.

CHORUS: SEND OUT, SEND OUT,
OH, SEND OUT THE GRADES TO BIG COMPANIES,
SEND OUT, SEND OUT,
THEY'LL ALL WANT A SCHOLAR LIKE ME!

TERRY BOLLING& THE WATT FIVE
COMP SCI DEPT, UNIV. OF MISSOURI, ROLLA
FROM CREATIVE COMPUTING, MAR/APR 1977

-- Courtesy of John "Med.Soohoo"

From chris Tue Apr 13 12:34:19 1982
To: auugn
Subject: From Norman Wilson

From ucbvax!cithep!norman Wed Apr 7 19:52:31 1982 netmail from usa
Subject: No close sometimes

Have you ever had the obscure problem that a device which tries to allow only one open at a time sometimes gets closed without its exclusion flag being cleared, as if the last close of the device didn't call the driver's close routine? If so (and even if not) you should move the two lines that set fp->f_flag and fp->fp_inode from after to before the code that calls itrunc in sys2/copen to avoid a possible race with the loop at the end of fio/closef. I have been looking for this one for at least a year!

NETMAIL NETMAIL NETMAIL NETMAIL NETMAIL NETMAIL NETMAIL NETMAIL NETMAIL NETMAIL

From pwbcc!andrew Tue Mar 16 16:07:05 1982 netmail from usa

I wrote this ballad a couple of weeks ago, and gave a premiere performance at USENIX with esquire!psl at the keyboard, and a chorus of hackers in the background. We were dubbed "Andy and the Free Inodes." The tune is the same as "Deep Elem Blues," recently covered by the Grateful Dead. Mail additions to floyd!trb.

It has about 1k verses to go, but, by popular demand, here it is,
The VMUNIX Blues:

VMUNIX Blues 21-Jan-82 by Andy Tannenbaum
(c) 1982 by Andy Tannenbaum
Profits to go to the Smithsonian Hacker Folk Music Archive.

When you bring up VMUNIX,
Just to have a little fun,
You'd better have your Western License,
When the Board of Regents come.

Oh, sweet mama, hacker's got the VMUNIX Blues,
Oh, sweet mama, hacker's got the VMUNIX Blues.

Once I met a pinstripe,
Knew his doubly nested do,
He logged in to VMUNIX,
Now his FORTRAN days are through.

Oh, sweet mama, ...

When you debug a C program,
Don't know what the hell to do,
Segmentation violation,
It's a structure pointer screw.

Oh, sweet mama, ...

It comes time to clean up shit work,
You type r m foo space star,
By the time you realize it,
You won't know where those files are.

Oh, sweet mama, ...

When you run on an eleven,
With a quarter meg of core,
And some loser cranks an nroff,
There ain't no response no more.

Oh, sweet mama, ...

Ar as at,
Bc dc adb,
Df du dd,
It don't mean a thing to me.

Oh, sweet mama, ...

When you argue with a netnews flamer,
Be prepared to lose,
Because that burnt out flamer's got,
The VMUNIX Blues,

From peteri Mon Mar 1 09:17:31 1982 netmail from elecvax
>From pwbcc!andrew Sat Feb 27 22:26:32 1982 netmail from usa

(a0457) r a PM-Lights 01-23 0411
?^PM-Lights,420<
^On The Light Side<

BOISE, Idaho (AP) - Idaho has introduced a new variety of animals to its 1982 fishing regulations - the ''wildwife.''

The breed was entered on state records when the Idaho Fish and Game Department printed regulations allowing Idaho taxpayers to donate income tax refunds to a fund promoting hunting of non-game animals.

The message advised sportsmen to ''Help Promote Non-Game Wildwife.''

The first 140,000 copies containing the typographical error were sent out before the rest of the 350,000-copy press run was corrected.

CASA GRANDE, Ariz. (AP) - It didn't make a dime of difference to Shirley Jean Kelly's lawyer how her former husband paid the divorce fees.

Truck driver Frank Kelly cleared the books Friday by paying the $354 tab in pennies - 35,450 of them. Kelly, 47, said he threw in an extra roll of 50 pennies just in case his count was off.

''We'll take them,'' said Shirley Jean's lawyer, Richard Clemons, when the pennies were pitched his way. He gave Kelly a receipt.

Kelley said he had been saving pennies since he was 19 and said: ''I still have an ample supply.''

It was his third marriage.

COSTA MESA, Calif. (AP) - Mention Vegemite to displaced Australians or New Zealanders and a dreamy, faraway expression may come over their faces as they recall a predilection similar to the American taste for peanut butter.

But until recently, most Australians or New Zealanders were as likely to find a kangaroo in the United States as Vegemite.

Now the Vegemite drought is over, according to two Costa Mesa importers who have 30,000 6-ounce jars of the stuff sitting in a warehouse waiting for a distributor.

Diana R. Todd and Jess Dines of Australasia Ventures hope not only to appease the Down Under transplants but also to convert the masses to the health and diet wonders of Vegemite, as applied to crackers, grilled cheese and meats, soup, stews, gravies, milk, grated carrots, raisins and chopped nuts and ''cooked mashed brains.'' Some claim Vegemite is the richest dietary source of vitamin B complex.

Ms. Todd, a native of Australia, has been marketing Australian products in this country for 13 years. But Vegemite is a real challenge. She often runs into attitudes like those mentioned by the Los Angeles Herald Examiner Friday.

''Some detractors say the creamy, tangy, dark brown yeast extract tastes like coal tar, axle grease or dried anchovy-flavored varnish,'' the paper said.

?AP-NR-01-23 0525EST<

AUUGN is produced by volunteers from the Australian Unix Users Group. To sustain the fine standard of journalism which our discriminating clientele have come to expect (or will soon, anyway), we solicit material from readers. This means YOU! Yes YOU! Send your material to the editors at the addresses given below. We prefer to be netsent the unformatted file, but hard copies are gratefully accepted and reproduced intact. Also, material should be in the public domain.

Now about that other thing. Money. We need to increase our readership, so get anyone you can to join the Australian Unix Users Group. For a mere $24 you will receive six issues of AUUGN over a period of one year. Please send your cheques in Australian currency. Do not send purchase orders. So mail your subscription fee to this address:

AUUGN
c/o Bob Kummerfeld
Basser Department of Computer Science
Madsen Building F09
University of Sydney
NSW 2006
Australia

Electronic corresponence should be addressed to:

auugn:basservax (on the SUN)
mhtsa!australia!auugn (in USA)