stichting mathematisch centrum

MC

AFDELING INFORMATICA IW 201/82 AUGUSTUS
(DEPARTMENT OF COMPUTER SCIENCE)

D.S.H. ROSENTHAL

UNIX AS A BASIS FOR CAD SOFTWARE

Preprint

kruislaan 413 1098 SJ amsterdam

Printed at the Mathematical Centre, 413 Kruislaan, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

CR Categories and Subject Descriptors: D.2.6 [Software Engineering]: Programming Environments J.6 [Computer Applications]: Computer-Aided Engineering — Computer-aided design

UNIX as a Basis for CAD Software

by

David S. H. Rosenthal

ABSTRACT

The suitability of the UNIX* operating system as a basis for constructing and using CAD software is assessed in the light of several years experience of using it for this purpose in the Edinburgh University Department of Architecture.

This paper was published in *Computer Aided Design* **14**(3), (May 1982).

* UNIX is a Trademark of Bell Laboratories.

1. Introduction

The UNIX operating system was written for the PDP-11 at Bell Labs in 1970-71[22, 23]. Although it has been widely used in universities since the mid-70s[17], until recently it has received relatively little attention elsewhere. Recent developments have dramatically changed this:-

- Several demonstrations of the system's portability[11, 19] have led to its becoming available on a wide range of hardware. The system is currently marketed for the IBM 370 architecture, the Perkin-Elmer 32XX, the VAX, the PDP-11 series, the BBN C/70, the Motorola 68000, and the Zilog Z8000.*
- The advent of single-chip processors capable of running the system with ease has reduced the hardware cost of entering the world of UNIX to about 8,000 pounds. Changes in licensing arrangements mean that a supported binary license is now available for about 2,000 pounds. A high-quality timesharing service at this price has proved very attractive.
- UNIX appears to be the best candidate for a *de facto* standard operating system for the 16-bit microprocessors, filling the rôle which CP/M has played for smaller machines.
- UNIX's reputation as a highly productive environment for programmers to work in has stood the test of much significant commercial usage, both inside[6] and outside[20] the Bell system.

The prospect of a powerful operating system becoming accepted over a wide range of hardware is of particular importance to the CAD community. Typical CAD programs interact with their host operating systems in complex ways; they address the peculiarities of physical terminals doing graphics, they maintain disk databases large enough to hit system limits, and access them in random and unpredictable ways. Finally, they are often large and demanding in response time; significant efforts frequently being invested in adapting their behaviour to use their host system efficiently.

* Implementations exist, or are underway, for many other machines, including the Univac 1100, the Harris /6[16], the Three Rivers Perq, and the IBM Series/1[27].

Even if a program of this kind, running on a manufacturer supplied operating system, has been written in a "standard" language, porting it between different systems will be a non-trivial task. An operating system available on a wide range of hardware providing the facilities required to construct CAD systems would greatly improve the portability and saleability of CAD systems. Developers and users could choose the most cost-effective hardware available at the time of purchase; systems would no longer become unduly expensive through being tied to a particular manufacturer's hardware.

This paper is not intended as an introduction to the concepts and facilities of UNIX. For this, the reader is referred to the original description in *Communications of the ACM*[22], to the July-August 1978 special issue of the *Bell System Technical Journal*, and to the paper by Kernighan and Mashey[14].

2. The UNIX Environment

The UNIX environment can be provided in one of two ways. The UNIX kernel can run directly on the bare hardware, using no manufacturer-supplied software at all, as it does on the PDP-11. Alternatively, if the manufacturer supplies a sufficiently adaptable "virtual machine" system, it can run as a virtual machine alongside others on top of the manufacturer's software. This technique has been used by, among others, Amdahl on VM/370.

A language standard, such as FORTRAN 77, specifies how an application may perform arithmetic, and open, close, read, and write files. Some other languages go a little further, specifying for example how an application may dynamically obtain and release memory. However, interactive CAD programs require a far wider range of services. They may need to manipulate terminal modes, change file access permissions, intercept signals, create and manage sub-processes, send and receive messages, and so on. A standard environment means that programs requiring services such as these need not sacrifice portability.

Of course, a standard environment in terms of system calls and utilities does not itself guarantee program portability; compilers are inevitably dependent on the underlying hardware, and may therefore be somewhat incompatible. Fortunately, considerable efforts have been devoted to providing UNIX with portable compilers. The kernel and all the utilities are written in C; among the utilities is a C compiler[12] designed to be easy to re-target. This should always be both available and efficient. The FORTRAN 77 compiler[7] uses the same code generator, and is thus ported with little extra effort. The Free University of Amsterdam have developed a portable Pascal system, based around a high-level intermediate code[24]. Porting this requires writing a relatively simple intermediate-code translator. Many other languages are available, though because the system's portability is fairly recent, most are not portable.

3. Basic Facilities for CAD

CAD applications implemented under UNIX may be regarded as collections of *processes* operating upon data stored in the *file system* and communicating with *terminals*. The facilities of the system will be discussed under each of these headings.

3.1. Processes

The UNIX environment encourages the construction of applications from collections of small programs communicating via files and pipes,* rather than as large, monolithic programs. Each user may have many (typically up to 12) simultaneously executing processes. Executing a new program is a frequent operation, critical to performance. It is accomplished by creating a new process, which then executes the new program and dies. Typical costs for this sequence are 0.14s (on a VAX11/780) and 0.43s (on a PDP11/34) in real time, representing between 0.10s and 0.26s of processor time, over and above the cost of the program.

* A pipe is an inter-process communication channel, which behaves as an anonymous FIFO file.

For each implementation of UNIX, there will be a limit on how much memory an individual process can address. Swapping systems, such as those for the PDP-11 and the Z8002, require the whole of the program and data to be in real memory while being run, and typically have limits corresponding to 16-bit addressing. On processors such as the PDP-11/23, which cannot distinguish between instruction (I) and data (D) accesses, the program and the data together are limited to 64K bytes. On processors such as the Z8002, which can distinguish I and D spaces, the program and the data are separately limited to 64K bytes each.

These limits have proved a major constraint, particularly on the non-I/D PDP-11s. As a result, a text overlay scheme has been developed at Berkeley[9] which permits programs somewhat larger than the 64K limit to be run, though enough physical memory for the entire program and data must still be available. The effect of the overlay scheme is shown in Figure 1.

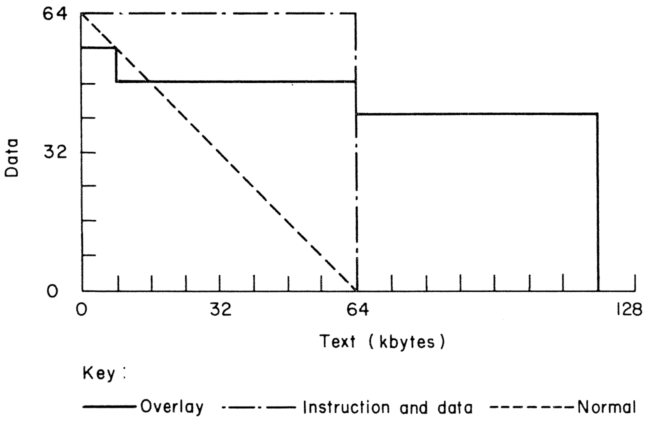

Figure 1. Data size versus *maximum text size*

The overlay technique could also be applied to an I/D processor, in which case data sizes up to 64K bytes could theoretically co-exist with text sizes up to 400K bytes. However, swapping programs of this size as a unit is quite impractical. Overlay schemes in which the inactive overlays are disk-resident are not often used; programs are divided into separate processes instead.

These problems, and the advent of the VAX, led the Berkeley team to adapt UNIX so as to exploit the virtual memory capabilities of suitable processors[3]. Virtual memory UNIX is normally configured to support processes with up to 6M bytes of program and up to 6M bytes of data on a VAX; extensions to support much larger processes are possible.

In practice, we have found the multi-process facilities of the system exceptionally useful. The ability, for example, to invoke the screen editor from within an application program intended to examine and alter textual information in a database, makes it very easy to provide a powerful and uniform user interface, and reduces the amount of code to be written and debugged. On the other hand, the limits on process address space have been a significant constraint. A simple drafting system[4], for example, had to be implemented as two processes.

3.2. File System

The UNIX file system is fairly conventional, but extremely simple. It consists of a rooted tree of files, each interior node of which is a directory (a file containing names and addresses of files), and each leaf of which is either a file or a directory. Each file is just a featureless, randomly addressable sequence of between 0 and 1,082,201,087 bytes. It has been observed that:-

> ".... a file is best described by the attributes it lacks.
>
> - There are no tracks or cylinders; the system conceals the physical characteristics of devices instead of flaunting them.
> - There are no physical or logical records or associated counts; the only bytes in a file are the ones put there by the user.
> - Since there are no records, there is no fixed/variable length distinction and no blocking.
> - There is no preallocation of file space; a file is as big as it needs to be. If another byte is written at the end of the file, the file is one byte bigger.
> - There is no distinction between random and sequential access; the bytes in a file are accessible in any order.
> - There are neither file types for different types of data nor any access methods; all files are identical in form.
> - There is no user-controlled buffering; the system buffers all I/O itself."*

This extreme effacement of all hardware features might be expected to result in poor performance from the file system, but measures are taken to avoid this. The kernel maintains a cache of recently used disk blocks in memory. On our system this cache is 25 blocks big, and has a typical hit rate of 75 to 80% (that is, only 1 in 4 or 5 I/O system calls actually results in disk I/O). This high hit rate is achieved by several tricks, the most important of which is *read-ahead.* If the kernel detects that a file is being read sequentially, it will pre-read into the cache the block ahead of the block being read, a very effective strategy.

Although access to large files is relatively efficient, the environment encourages the use of many small files. Figure 2 is a histogram of the sizes of nearly 7000 files on our machine, showing that the most popular file size is less than 512 bytes, and 80% of files contain less than 5120 bytes. On the other hand, Figure 3

* Reproduced from [14] by permission of the publisher, John Wiley and Sons, Chichester, U.K.

shows that these small files occupy only a small proportion of the space. Of course, dividing data into many small files means that open and close operations are very common, and their efficiency becomes critical. Opening and closing a file typically costs 71ms processor time (125ms real time) on a PDP11/60.

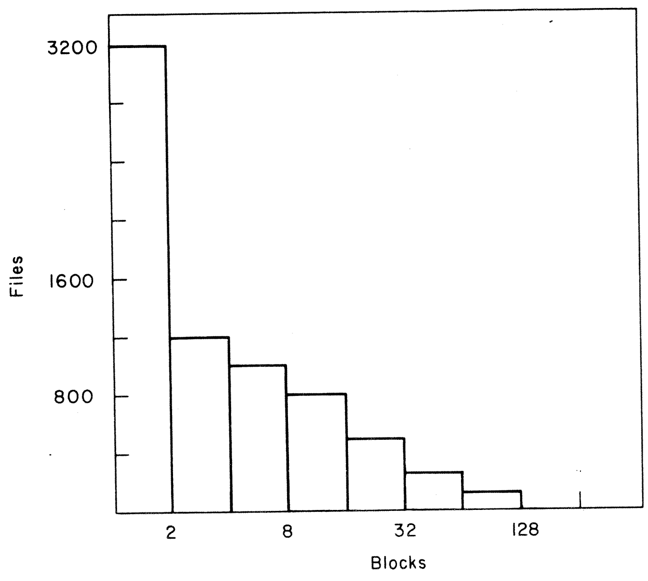

Figure 2. File size histogram

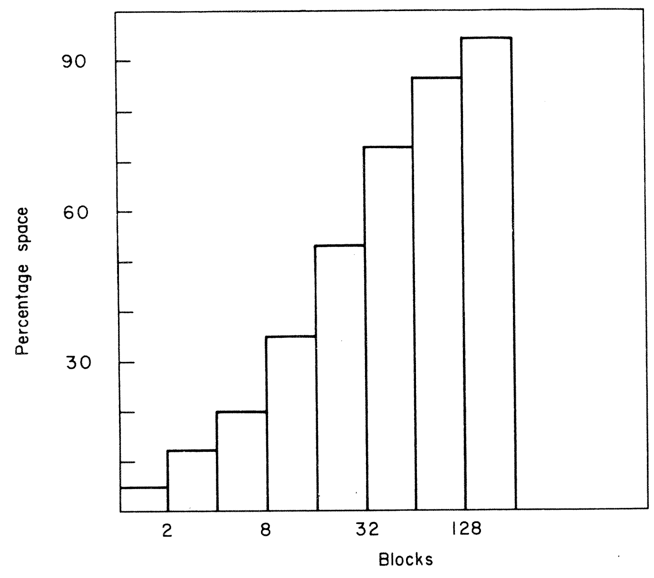

Figure 3. Cumulative space versus *file size*

Because the file system accesses large numbers of small files efficiently, and because the same file may appear in different directories under different names (multiple *links*), it is possible to use the file system to provide multiple access paths to application data. For example, a single file containing information on an individual part might have links from directories containing:-

- all parts used in a particular assembly, under their part numbers,
- all parts stored in a particular area, under their bin numbers,
- all parts supplied by a particular supplier, under his reference numbers,
- all parts on order from a particular supplier, under their order numbers,

and so on. Data stored in this way is easily accessible to programs written in the Shell (the language of the command interpreter), which is an extremely quick way to create new applications. On a typical system, 13% of the globally accessible commands are command files of this type. The proportion of personal commands implemented this way is much higher.

Two aspects of the file system's organisation cause performance problems on large (100s of Mb) disks. When data is appended to files, new blocks are simply acquired from the head of the appropriate free list; no special effort is made to keep the blocks of a file close together. In addition, the disk mapping information is in a single contiguous area at the start of the disk. Thus, on large physical disks head movement becomes excessive. The problem can be alleviated by treating a large physical disk as a number of smaller logical volumes, but moving parts of the file system to another volume when one volume's free space is exhausted is operationally inconvenient.

An application which required more efficient access to disk storage than that provided by the file system could use the so-called 'raw' devices. These by-pass the disk cache, transferring directly between the disk and the user process' address space. They are used in exactly the same way as normal files, except that they can only be read and written in units of the block size. Using raw devices, and the appropriate magic numbers, disks can be read or written in complete tracks or cylinders. Because raw devices are just files, if data is read in block size units the decision on their use can be postponed to run-time.

Although these facilities provide the opportunity to design an application-specific disk storage system, even major database applications such as the INGRES relational database manager have not taken it. In retrospect[25], however, the INGRES team believe that it would have been wise to do so. Using virtual memory UNIX, sparse access to large files (e.g. CAD databases) can be made more efficient by mapping them into the process' virtual memory. Use of this technique will increase as it becomes more widely available.

We have used the file system to implement various data management schemes for textual, numeric, and geometric information, including one based on Applied Research of Cambridge's BOS[1]. In all cases the simple and uniform interface has made writing and maintaining the programs easy, and the file system's performance has been adequate.

3.3. Terminal I/O

Terminal I/O uses the normal file system calls, is full-duplex, and can be as fast as the hardware will permit.* Programs designed to read and write files can read and write terminals with no special preparation, and vice versa. The I/O redirection facilities of the shell make this usual.

Characters typed at the terminal are normally accumulated until an end-of-line character, and other special characters are defined to perform delete-line, delete-character, interrupt process and end-of-file functions. Characters sent to the terminal may be subject to an X-ON/X-OFF protocol. A system call is provided to change all these special character assignments, to change line speeds and delays, and to set various modes on a dynamic, per-line basis.

* DMA asynchronous line multiplexers are an advantage, if available. Without them, transmitter interrupt service alone has been measured at 10% of an 11/70.

The modes available include one which passes all 8 bits, and one which wakes the reading process on every character. Another mode provides upper-to-lower case mapping. Case differences are significant for UNIX; using an upper-case only terminal is possible but inconvenient.

We have used the terminal facilities to build interactive graphics systems driving various terminal types, spoolers for various plotters and printers, and computer-to-computer links. Constructing these has been significantly easier than our experience with other systems. The complete control over special characters, and the acceptable efficiency of character-at-a-time I/O have been especially useful.

4. The System in Use

Although UNIX as a productive environment for programming has been discussed exhaustively[14, 10], its characteristics as an environment for routine production computing have received much less attention.

The routine of administering and running a UNIX installation is not onerous. If the system has been properly organised initially,† all the routine tasks are performed by shell scripts run at startup, shutdown, and at regular intervals between. On our system these perform file system consistency checking, incremental file dumps, and system performance monitoring. Others add usage accounting, disk quotas, security audits, etc.

Another system administration task for which the system provides assistance is that of introducing new users to the facilities they need. A computer-aided instruction system called Learn[13] is provided, capable of monitoring user's interactions with programs and, based upon their actions, walking them through a network of scripts providing graded

† The assistance of a "guru" is essential at this stage.

amounts of help. Scripts are provided to teach the editor and other basic facilities; these can be used as examples for creating new scripts.

A properly set up UNIX is almost as reliable as the hardware. However, the effects of a crash used to be a problem. Although the system rarely crashed so as to lose files completely, it usually left file systems in an inconsistent state. The tools available to repair damaged file systems were effective in skilled hands, but dangerous otherwise. Work since the release of the 7th Edition has both improved the crash-resistance of the file system, and provided a new file system check and repair program[15]. This can be used interactively, but is quite safe to use automatically on re-booting the system.

UNIX administers its resources through a number of internal tables, whose sizes are set at system generation time. It is possible to make these tables so big that user processes cannot exhaust the space in them, but this is wasteful of address space and physical memory, and risks are frequently taken on smaller configurations. The risk is that programs may occasionally fail through no fault of their own during times of heavy load, because they are unable to acquire the resources they need. This is rationalised as either a good argument for upgrading the configuration, or as a useful deterrent to anti-social behaviour (such as running resource-intensive programs during periods of heavy load), depending on one's point of view.

Experience of running the system in a hostile environment* indicates that, with minor modifications, it is capable of resisting determined and informed attempts at penetration. However, the price of such security is eternal vigilance, and few installations are willing to pay it. A level of security sufficient to resist all but the highly skilled is obtained through simple administrative attention to passwords, default file permissions, etc.

Our experience of running the system has been atypical, we are one of a tight group of about a dozen UNIX systems, with high levels of expertise.

5. The Future

5.1. Problems

We have identified a number of inadequacies with current UNIX implementations as viewed from the perspective of CAD. These include:

- The limited per-process address space.
- The performance of the file system on large disks.
- Interprocess communication facilities, particularly to support server processes and network access.

In addition, Stonebraker, from the standpoint of DBMS support, has identified other problems[26]. His criticisms, most of which apply to operating systems in general, rather than just to UNIX, include:

- The inappropriate nature of the LRU replacement strategy used in the buffer cache, when supporting databases.
- The inappropriate nature of the read-ahead strategy used in the buffer cache, when supporting databases.
- The need to selectively force blocks out of the cache to provide correct crash recovery.
- The performance problems of scattering the blocks of a file.
- The expense of switching between processes.

Of course, all the problems of file I/O for databases could be resolved by designing database-specific file systems on top of the 'raw' devices, which bypass the cache completely. However, it would be more satisfactory if the standard file system provided adequate performance.

* Australian computer science undergraduates.

5.2. Developments

The most important constraint on existing small systems is the lack of process address space. Although at present virtual memory UNIX is only available on fairly expensive computers, recent announcements from the chip-makers encourage the belief that small computers running the system are not too far off. The changes to come in the MC68000 microcode, and the forthcoming Z8003 and Intel 432 processors, should solve the problem.

It is anticipated that single-user workstations, running UNIX on one of these processors, together with an 8-inch Winchester disk, a high-performance raster display, a local-area network interface, and a digitising tablet, will be available shortly for about $15,000.

The problem of maintaining the performance of the file system on very large disks is being addressed. Techniques for adding new blocks to files which are near the existing blocks, and for distributing the disk mapping information through the space mapped, should both reduce the average length of seeks. The alternative approach, of replacing or augmenting the standard file system by an extent-based system, has been tried by a number of groups[18].

The problems caused by caching strategies inappropriate for particular processes can be addressed in the same way that Berkeley have accommodated processes for which the standard page replacement strategy is inappropriate. They provide a system call, used for example by LISP during garbage collection, by means of which a process may advise the paging system of its anticipated behaviour. Analogous system calls could be added to advise as to suitable blocks to read-ahead, and suitable blocks to flush from the cache.

Although UNIX has been connected satisfactorily to many networks, including the ARPANET[5], X-25, Cambridge Rings, and various experimental networks[8], the experiences of the various groups involved have led to a re-evaluation of a number of basic system concepts[21]. Several efforts are underway to re-design the inter-process and terminal I/O capabilities as an integrated unit, allowing network and multiple virtual terminal support as upwards compatible extensions of current system facilities. The experimental system at Bell Labs indicates that these permit a group of linked UNIX systems to be presented to the user as a single resource; operations can be performed wherever the user finds convenient.

The excessive cost of switching between processes is an inevitable result of the power of the process concept in a multi-user multi-purpose system. Processes have a large quantity of context information, and traditional processors lack hardware to support it. However, processor design is now being affected by knowledge about the software they are going to run. Perhaps the most important effect of the knowledge that they will run UNIX is the provision, for example on the BBN C/70[2], of hardware support for rapid context switching.

6. Conclusions

Our experience indicates that the UNIX environment provides all the basic facilities required by CAD systems, on a range of machines from 16-bit microprocessors to large mainframes. Some constraints imposed by the smaller processors have been irritating, but on more modern processors they have been removed.

7. Acknowledgements

This work was supported by Science and Engineering Research Council grant GR/A80341. Special thanks are due to my colleagues Jim McKie and Fernando Pereria.

REFERENCES

[1] ARC, *BOS Infrastructure Software - A Short Technical Description,* Applied Research of Cambridge, Ltd. (1978).

[2] BBN, *C/70 Hardware Reference Manual,* BBN Computer Co., Cambridge, Mass. (March 1981).

[3] O. Babaoğlu, W. Joy, and J. Porcar, *Design and Implementation of the Berkeley Virtual Memory Extensions to the UNIX Operating System,* Computer Science Division, University of California, Berkeley, California.

[4] A. Bijl and J. Nash, "Progress on Drawing Systems," *Computer Aided Design* **13**(6), pp.351-358 (November 1981).

[5] G. L. Chesson, "The Network UNIX System," *Operating Systems Review* **9**(5), pp.60-66 (1975).

[6] T. A. Dolotta, R. C. Haight, and J. R. Mashey, "UNIX Time-Sharing System: The Programmer's Workbench," *Bell Sys. Tech. J.* **57**(6), pp.2177-2200 (1978).

[7] S. I. Feldman, "Implementation of a Portable FORTRAN 77 Compiler Using Modern Tools," *ACM SIGPLAN Notices* **14**(8), pp.98-106 (Aug 1979).

[8] A. G. Fraser, "Datakit - A Modular Network for Synchronous and Asynchronous Traffic," *Proc. ICC* (June 1979).

[9] C. Haley and W. N. Joy, "Running Large Text Processes on Small UNIX Systems," Computer Systems Research Group, Dept. EECS, University of California, Berkeley, California.

[10] E. L. Ivie, "The Programmer's Workbench — A Machine for Software Development," *Comm. Assoc. Comput. Mach.* **20**(10), pp.746-753 (October 1977).

[11] S. C. Johnson and D. M. Ritchie, "UNIX Time-Sharing System: Portability of C Programs and the UNIX System," *Bell Sys. Tech. J.* **57**(6), pp.2021-2048.

[12] S. C. Johnson, "A Portable Compiler: Theory and Practice," *Proc. 5th ACM Symp. on Principles of Programming Languages*, pp.97-104 (January 1978).

[13] B. W. Kernighan and M. E. Lesk, *LEARN — Computer-Aided Instruction on UNIX (Second Edition),* Bell Laboratories, Murray Hill, New Jersey (January 1979).

[14] B. W. Kernighan and J. R. Mashey, "The UNIX Programming Environment," *Computer* **14**(4), pp.12-24, (adapted from *Software — Practice & Experience* **9**(1) January 1979 pp1-16) (April 1981).

[15] T. J. Kowalski, *FSCK — The UNIX File System Check Program,* Bell Laboratories, Murray Hill, New Jersey (1980).

[16] S. J. Leffer, *A Detailed Tour through the /6 Portable C Compiler,* Dept. of Computer Engineering, Case Western Reserve University, Cleveland, Ohio (1980).

[17] J. Lions, "Experiences with the UNIX Timesharing System," *Software — Practice and Experience* **9**(9), pp.701-709 (Sept 1979).

[18] H. Lycklama and D. L. Bayer, "The MERT Operating System," *Bell Syst. Tech. J.* **57**(6) (1978).

[19] R. Miller, "UNIX — A Portable Operating System?," *Operating Systems Review (ACM/SIGOPS)* **12**(3), pp.32-37 (June 1978).

[20] I. R. Perry, "UNIX From the Point of View of a Commercial User," *European UNIX User Group Newsletter*(7), pp.28-31 (July 1980).

[21] R. F. Rashid, "An Interprocess Communication Facility for UNIX," CMU-CS-80-124, Computer Science Dept., Carnegie-Mellon University, Pittsburgh, Pennsylvania (February 1980).

[22] D. M. Ritchie and K. Thompson, "The UNIX Time-Sharing System," *Comm. Assoc. Comput. Mach.* **17**(7), pp.365-375 (July 1974).

[23] D. M. Ritchie, "The Evolution of the UNIX Time-Sharing System," in *Proc. of Symp. on Language Design and Programming Methodology, Sydney, Australia*, ed. J. M. Tobias, Springer-Verlag, Berlin (September 1979).

[24] J. W. Stevenson, *Pascal-VU Reference Manual,* Wiskundig Seminarium, Vrije Universteit, Amsterdam (March 1980).

[25] M. Stonebraker, "Retrospection on a Database System," *ACM Trans. on Database Systems* **5**(2), pp.225-240 (June 1980).

[26] M. Stonebraker, "Operating System Support for Database Management," *Comm. Assoc. Comput. Mach.* **24**(7), pp.412-418 (July 1981).

[27] P. Jalics, Cleveland State University, USENIX meeting, Boulder, Colorado (February 1980).

MC NR

35223