THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 57, No. 6, July-August 1978
Printed in U. S. A.

UNIX Time-Sharing System:

RBCS/RCMAS—Converting to the MERT Operating System

By E. R. NAGELBERG and M. A. PILLA
(Manuscript received February 3, 1978)

The paper presents a case history in applying the MERT executive to a large software project based on the UNIX system. The work illustrates some of the basic architectural differences between the MERT and UNIX systems as well as the problems of portability. Emphasis is on matters pertaining to software engineering and administration as they affect development and support of a manufactured product.*

I. INTRODUCTION

The Record Base Coordination System (RBCS) and Recent Change Memory Administration System (RCMAS) are two minicomputer-based products designed to carry out a record *coordination* function; i.e., they accumulate segments of information received from various *sources* on different media, filter, translate, and associate related data, and later transmit complete records to downstream *user* systems, also on an assortment of media and according to various triggering algorithms. The overall objective of record coordination is to assure that information stored and interchanged among a variety of related systems is consistent and accurately reflects changes that must continually occur in the configuration of a large, complex telecommunications network. To perform this function, RBCS and

* UNIX is a trademark of Bell Laboratories.

RCMAS both provide various modes of I/O, data base management, etc., and each utilizes a rich assortment of operating system features for both software development and program execution.

The RBCS project was initially based on the UNIX system,[1] making use of its powerful text processing facilities, the C language compiler,[2] and the program generator tools Yacc and Lex.[3] However, after a successful field evaluation, but just prior to development of the production system, it was decided to standardize using the MERT/UNIX* environment.[4] To a very large extent, this decision was motivated by the genesis of a second project, RCMAS, which was to be carried out by the same group using the same development facilities as RBCS, but which had much more stringent real-time requirements. The additional capabilities of the MERT executive, i.e., public libraries and powerful inter-process communication using shared segments, messages, and events, were attractive for RCMAS and, even though the RBCS application did not initially utilize these features, the MERT operating system was adopted here as well for the sake of uniformity.

The purpose of this paper is to present what amounts to a case history in transporting a substantial application, RBCS, from one operating system environment, the UNIX system, to another, the MERT/UNIX system. It is a user's view in that development and ongoing support for the operating system and associated utilities were carried out by other organizations. Of course, these programs were also undergoing change during the same period as the RBCS conversion. On the one hand, these changes can be said to confound the conversion effort and distort the results, but on the other hand a certain level of change must be expected and factored into the development process. This issue is referred to later as an important consideration in determining system architecture.

The paper begins with a discussion of the reasons for choosing a UNIX or MERT/UNIX environment, followed by analysis of the decision to utilize the MERT executive. The transition process is described, and a section on experience in starting with the MERT system, for purposes of comparison, is added. Throughout, the emphasis is on matters pertaining to software engineering and administration as they affect development and support of a manufactured product.

*MERT executive, UNIX supervisor program.

II. WHY THE UNIX AND MERT/UNIX OPERATING SYSTEMS?

With the exception of the interprocess communication primitives, the UNIX and MERT operating systems appear to the user as fairly equivalent systems. In large part, this is due to (*i*) the implementation of the UNIX system calls as a supervisory process under the MERT executive and (*ii*) the ability to use existing C programs such as the UNIX editor and shell with no modifications. Throughout this paper, unless emphasis of a particular MERT/UNIX feature is required, the phrase "UNIX operating system" will be used to mean the stand-alone UNIX operating system as well as the UNIX supervisor under MERT.

The facilities which the UNIX system provides for text processing and program development, the advantages of a high-level language such as C, and the power of program generators such as Yacc and Lex are described elsewhere in this issue. All these were factors in the final decision. However, the most compelling advantage of the UNIX system was the shell[5] program, which permitted the very high degree of flexibility needed in the RBCS project.

Because of the important role of field experience in the design process, developing a system such as RBCS or RCMAS is difficult, since a complete specification does not exist before trial implementation begins. Recognizing this fact, the developing department decided to make heavy use of the UNIX shell as the *primary* mechanism for "gluing together" various utilities. Use of the shell allows basic components to be bound essentially at run-time rather than early in the design and development stages. This inherent flexibility permits the system designer to modify and/or enhance the product without incurring large costs, as well as to separate the development staff into (*i*) "tool" writers, i.e., those who write the C utilities that make up the "gluable" modules and (*ii*) "command" writers who are not necessarily familiar with C but do know how a set of modules should be interconnected to implement a particular task.

In actual practice, some overlap will exist between the two groups if for no other reason than to properly define the requirements for the particular C utilities. The RBCS and RCMAS developments followed this approach and the evaluations of the projects overwhelmingly support the ease of change and administration of features. The response of the end users to the RBCS field evaluation system, in particular, has been most encouraging. In fact, they have written a few of their own shell procedures to conform more easily to local methods without having to request changes to the standard product.

In addition to the shell, heavy use of both Yacc and Lex was made to further reduce both projects' sensitivity to changes of either design or interface specification. For example, some processing of magnetic tapes for RBCS has been accomplished with parsers that produce output to an internal RBCS standard. For RCMAS, various CRT masks are required for entry and display of important information from its data base. An example is shown in Fig. I. These masks, being character strings tailored to the format of the specific data to be entered or displayed, lend themselves very naturally to design and manipulation using Yacc and Lex program generators. This allows rapid changes (even in the field) to the appearance of the CRT masks without incurring the penalty of modifying the detailed code for each customized mask.

Documentation for manufacture and final use was provided using the nroff[6] text formatting system. In addition, to make the system easier to use and to obtain uniformity of documentation, the UNIX form[7] program was utilized to provide templates for the various documentation styles.

With two development projects, short deployment schedules, and limited computer resources in the laboratory, it was necessary to *develop*, *test*, and *manufacture* software on the same machine. The software engineering required to support such simultaneous operations would have been difficult, if not impossible, without the various UNIX features.

III. WHY THE MERT SYSTEM?

If the UNIX operating system performed so well with its powerful development tools and flexible architecture possibilities, why, then, was RBCS converted from the UNIX to the MERT operating system and RCMAS directly developed under the MERT operating system? To answer this question, it is important to examine the underlying software engineering and administration problems in a project such as RBCS.

The organization responsible for end-user product engineering and design for manufacture should regard themselves *as users of an operating system,* with interest centered around the application. Once end-user requirements are defined, architectural and procedural questions should be resolved at all levels (*i*) to minimize change and (*ii*) to avoid the propagation of change beyond the region of immediate impact. Once the development is under way, changes of

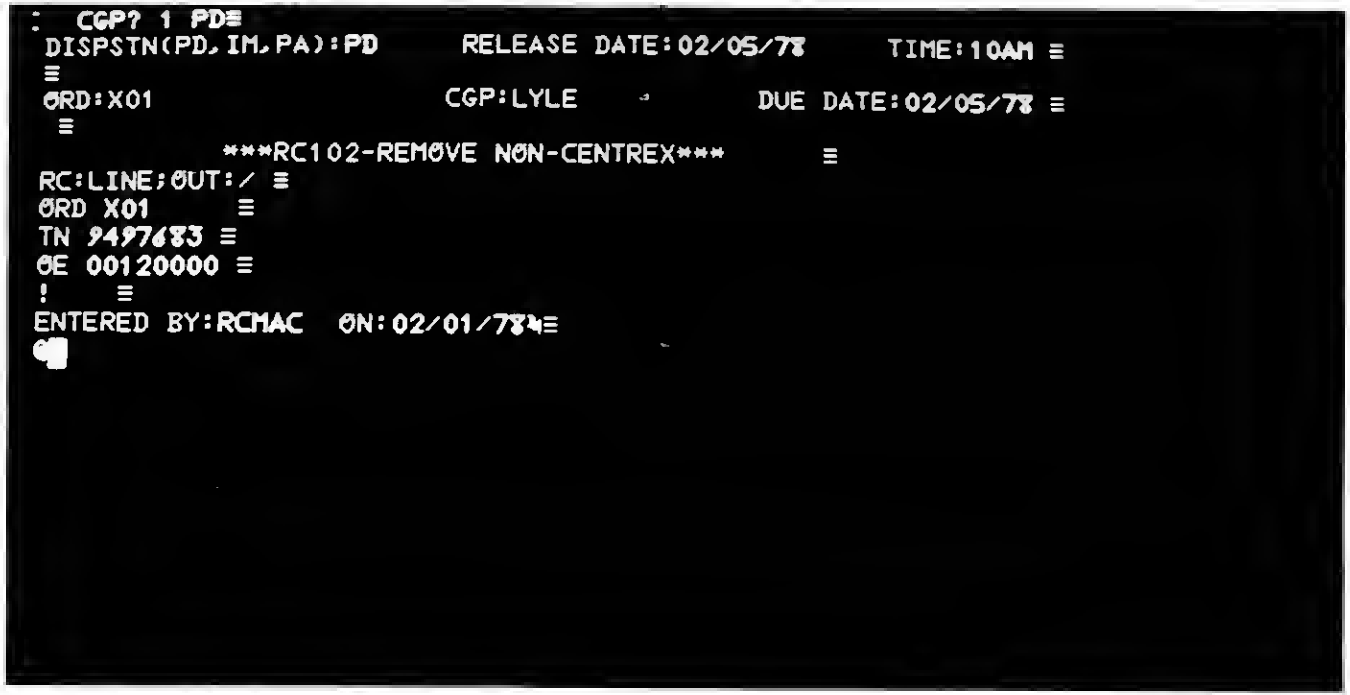

Fig. 1—RCMAS CRT mask.

any sort are undesirable if adequate administrative control is to be exercised, schedules met, and the product maintainable.

The nature of the RBCS project, however, necessitated changes to the UNIX operating system in three important areas: (*i*) a communications interface to support Teletype Corporation Model 40 CRTs, (*ii*) multifile capability for magnetic tape (e.g., standard IBM-labeled tapes), and (*iii*) interprocess communication for data base management. It was the problem of satisfying RBCS requirements in these areas under an ever-increasing rate of modifications to the "standard" UNIX system that led to a consideration of the MERT operating system.

Figure 2 graphically illustrates the problem of making slight modifications to the executive program of the UNIX operating system, especially in the area of input/output drivers. Because the UNIX system, including all drivers, is contained in a single load module, *any* modifications, even those at the applications interface, require careful scrutiny and integration, including a new system generation followed by a reboot of the system. On the other hand, referring again to Fig. 2, modifying a driver in a MERT environment is considerably easier because of the modular architecture of the MERT executive. In particular, the MERT executive is divided into several components, which can be administered and generated separately, frequently without a reboot of the system.

The experience over an 18-month period with the MERT operating system is that no RBCS or RCMAS driver was affected by changes to the MERT system. RBCS modifications to the magnetic tape and communications drivers were frequently tested *without* generating another system and, quite often, without rebooting. The advantages of not having to regenerate or reboot become obvious after a few iterations of change.

Furthermore, as shown in the encircled area of Fig. 2, it should be feasible, ultimately, for a project such as RBCS or RCMAS to receive the UNIX supervisor, MERT file manager, and MERT executive as object code rather than as source code. Both RBCS and RCMAS change only sizing parameters in these modules; such parameters could be administered after compilation, thus freeing a project from having to track source code for the major portions of the operating system.

With regard to interprocess communication, the translation and data-base management aspects of RBCS place a heavy strain on any operating system, but on the UNIX system in particular. For example, since the UNIX system was designed to provide a multi-access,

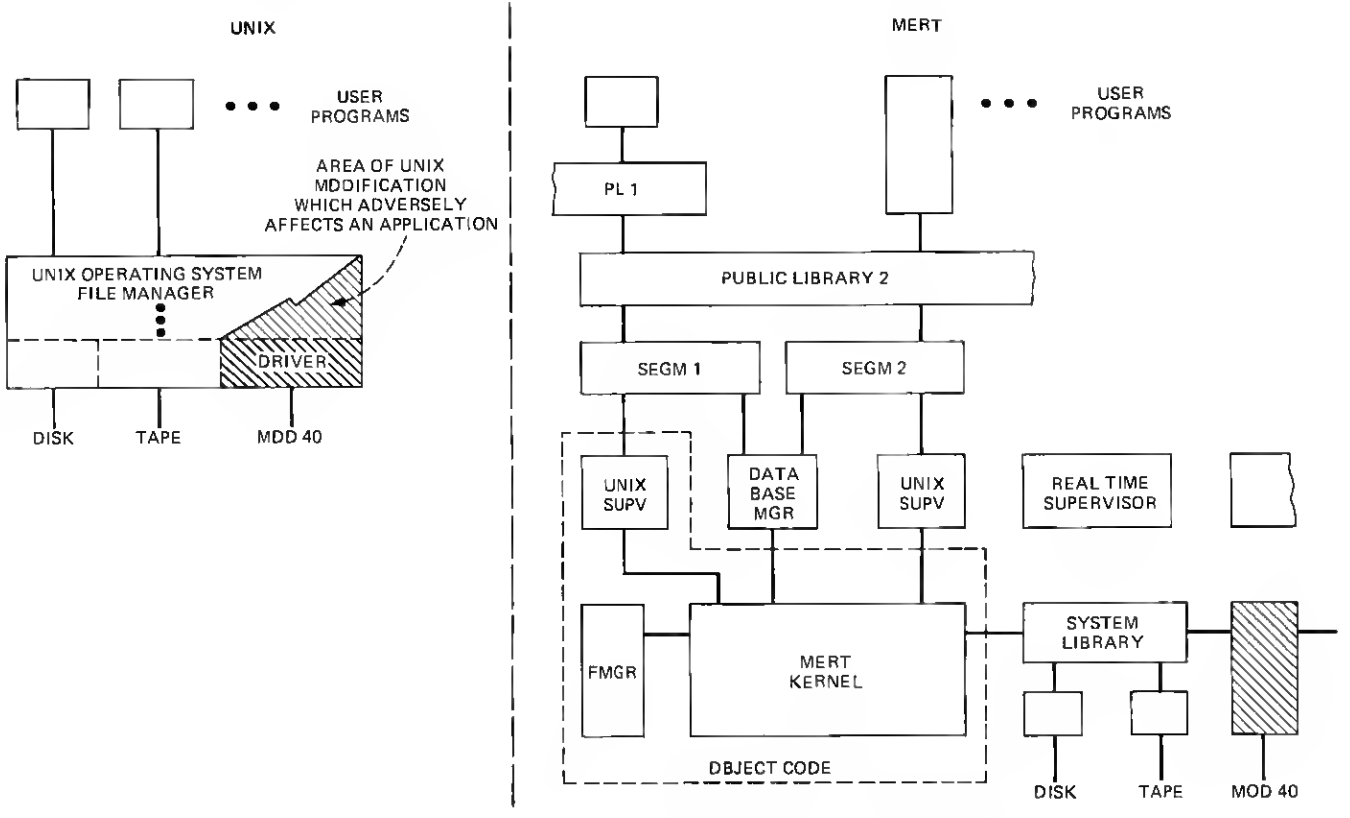

Fig. 2—UNIX™ and MERT executive architectures with possible application interfaces.

time-sharing environment in which a number of independent users manipulate logically separate files, a large body of common code, such as a data base manager, could only be implemented via a set of subroutines included with each program interacting with the data base. With such a structure, no natural capability exists to prevent simultaneous updates, and special measures become necessary. In general, it is difficult to allow any asynchronous processes (nonsibling related) to communicate with each other or to synchronize themselves without modifying the operating system or spending inordinate amounts of time with extra disk I/O.

Several measures were taken to try to overcome these limitations. Where sibling relationships existed by virtue of processes having been spawned by the same shell procedure or special module, UNIX pipes could be used, but these are quite slow in a heavily loaded system. Where nonsibling related processes were involved, specialized file handlers had to be used since large amounts of shared memory (within the operating system) were out of the question. It is important that the shared memory be contained in *user* space since different applications, at different points during processing, may place inordinate demands on a system-provided mechanism for sharing memory. This point is discussed further in the next section.

After considerable experience with the RBCS field evaluation, the time came to incorporate what had been learned and design a production system. It became quite clear that interprocess communications was our primary bottleneck and that this would be even more the case for RCMAS. An operating system was needed that could (*i*) support better interprocess communication than the UNIX operating system, (*ii*) support feature development (modifications to the operating system) without tearing into the kernel of the system or unnecessarily inconveniencing users by system generations, reboots, etc., and (*iii*) still provide the powerful development tools of the UNIX system. The MERT operating system met these three criteria quite satisfactorily.

IV. CONVERTING TO THE MERT OPERATING SYSTEM

It was essential in converting RBCS from the UNIX operating system to the MERT operating system to make as few application changes as possible; RBCS was already running so that only modifications necessary for enginering the production system, as opposed to a field evaluation, were incorporated. Confounding this, however, was a desire to upgrade to the latest UNIX features such as

(*i*) the latest C compiler, (*ii*) "standard" semaphore capability, (*iii*) "standard" multifile magnetic tape driver, and (*iv*) the Bourne shell.[5] There was also, of course, a desire to investigate and exploit those MERT features that were expected to yield performance inprovements.

The conversion was carried out over a period of 3 to 4 months by a team of three people. As expected, the major changes took place in the I/O routines, including (*i*) the CRT driver, (*ii*) the CRT utility programs, (*iii*) the magnetic tape utilities, and (*iv*) the RBCS data base manager, which allocated physical disk space directly. The procedure was straightforward and algorithmic, albeit a tedious process. Some manual steps were required, so that the conversion was not totally mechanized, but since the four basic subsystems mentioned above were edited, tested, and debugged in only 3 to 4 months, the effort was considered minimal.

At this point, an attempt was made to convert the remainder of the RBCS data base manager subsystem to the MERT public library feature.[4] Under the UNIX operating system, the RBCS data base manager routines were link edited into a substantial number of application utilities at compile time. Any changes required approximately two weeks for recompilation of the affected programs (the subroutine dependencies were kept in an administrative data base). It was felt, during the conversion planning effort, that the most common file manager routines were taking an inordinate amount of space; approximately 10 to 12K bytes duplicated among on the order of 100 routines; each with different virtual addresses for these common routines. The MERT public library feature appeared to solve the common space problem by allowing the most frequently used routines to be placed in a shared text segment. As of this writing, the basic system without the public library feature has been completed. Work is still underway to convert the RBCS data-base manager routines to the new MERT public library format and should be completed in a few months.

With the confidence gained by the initial success, work proceeded on obtaining compatibility with the latest version of the C language compiler, on the standard I/O library, and on engineering the manufacture of the production RBCS system.

The program which proved most difficult to convert was the "transaction processing" module. This large, complex body of code is responsible for (*i*) restricting write-access to the data base manager to avoid simultaneous updates and (*ii*) maintaining internal consistency between successive states of the numerous RBCS files.

These functions require considerable interaction with other shell procedures, generally on an asynchronous basis. Since the system was written under the UNIX file manager, the transaction processing module carried out this interaction through specially created files with a large number of "open" and "close" operations, characteristic of a time-sharing system. The MERT file manager restricted the number and frequency of these operations with the result that considerable effort was necessary first to analyze the problem and then to carry out the necessary design changes.

V. STARTING WITH THE MERT OPERATING SYSTEM

Probably the most important reason for using the MERT operating system, in addition to the above-mentioned architectural advantages, is the interprocess communication capabilities; in particular, shared segments, public libraries, and guaranteed message integrity.[4]

In Fig. 3a, the first module represents a "data base read" module, the second a "CRT interactive" package, the third a "data verification" module, and the fourth a "data base write" module. Fig. 3a illustrates the necessary pipeline flow of data using the UNIX "pipe" mechanism via the shell. Reverse flow of data (from cooperating modules, for example) is not possible at the shell level, and control flow is difficult without some form of executive. Furthermore, in a heavily loaded system, the pipes degenerate rather quickly to extra disk I/O; the result is a total of 4 reads and 4 writes for a simple data base transaction.

A typical transaction for the architecture illustrated in Fig. 3a would be for a keyboard query (module 2) for a particular data base record (module 1) to be displayed (module 1). Following local modification of the record using CRT masks, a request to send the record to the data base (module 4) is made. Before sending the record to the data base manager, the sanity check process (module 3) verifies the "within record" data integrity for such violations as nonnumerical data or illegal data. If the data check is unsuccessful, then module 2 should be activated to blink the fields in error as indicated by data obtained from module 3. Control flows alternately between modules 2 and 3 until either the record passes the data integrity check or local management overrides the process for administrative purposes. Only at that time would the record proceed to the data base write process (module 4). With a shell syntax, it is only possible for data to flow left to right; reverse flow requires files specifically allocated for that purpose.

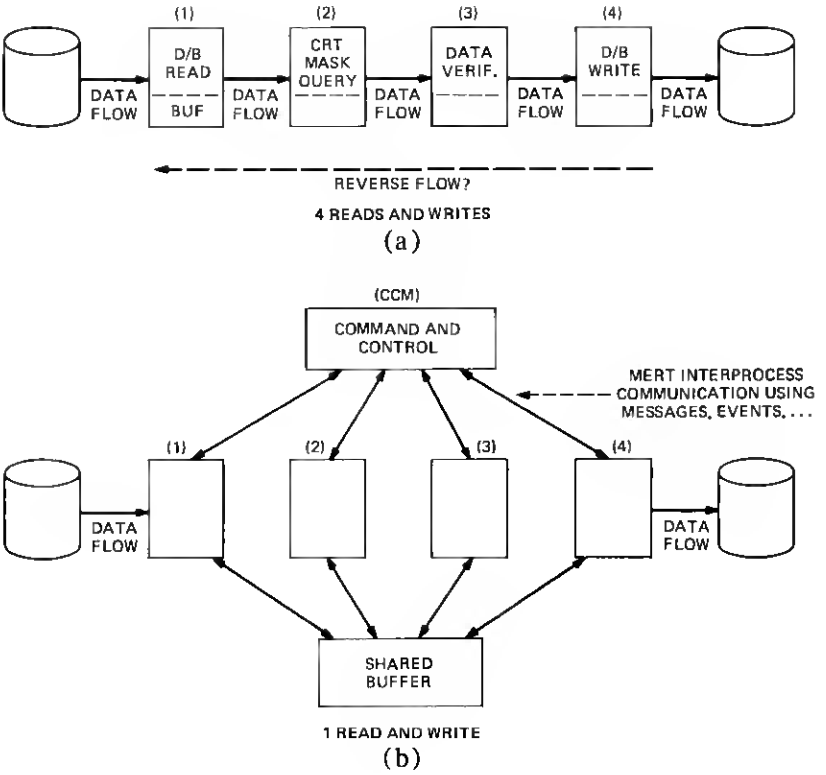

Fig. 3—(a) UNIX™ shell architecture. (b) MERT shell architecture.

Figure 3b illustrates the MERT approach with a single shared-data segment used for a minimum 1 read and 1 write. Each cooperating process is shown with a virtual attachment to the shared data segment indicating access to the data without disk I/O. Thus, reverse flow of data is accomplished by each process writing into the shared segment at any given time. Flow of control and process synchronization is accomplished in the example shown by the upper process called a "Command and Control Module" (CCM), in RCMAS. The shared data segment can be as large as 48K bytes in *user* space with MERT support provided so that sibling-related or even nonsibling-related processes can be adequately interfaced or isolated as the case requires *without* system modifications on our part.

The large user segment size allows individual RBCS or RCMAS transaction data to be supported with a minimum of disk I/O. The message capability, along with the segment management routines, allows

the data to *remain* in a segment; the processes modify the *single* copy of the data instead of passing the data around through pipes or files, as with the UNIX operating system implementation shown in Fig. 3a.

Rather than write an elaborate C module for the "Command and Control" process of RCMAS, the latest version of the shell written by S. R. Bourne[5] was extended to include the MERT interprocess communications primitives.[8] In particular, the "Command and Control Module" for RCMAS has been implemented entirely by means of one master shell procedure and several cooperating procedures. This has the advantage of easier readability to nonprogrammers, flexibility in the light of frequent changes, and ease of debugging (since any shell procedure can be run interactively from a terminal, one line at a time). Measured performance to date has not indicated any penalty great enough, from a time or space viewpoint, to necessitate rewriting the shell procedures as C modules.

It is clear from observations of RCMAS performance and discussions with interested people that considerably more flexible architectures are possible than the simple "star" network illustrated in Fig. 3b. However, it was felt that such a simplified approach was necessary to retain administrative control during the initial design and implementation stages until sufficient familiarization was achieved with asynchronous processes served by the elaborate MERT interprocess primitives.

VI. ACKNOWLEDGMENTS

The work of converting the RBCS project from the UNIX to the MERT operating system fell primarily on D. Rabenda, J. P. Stampfel, and R. C. White, Jr. The task of coordinating the design work for the RCMAS project was the responsibility of N. M. Scribner. Throughout the RBCS and RCMAS developments, especially under the MERT operating system, D. L. Bayer and H. Lycklama were most helpful with the operating system aspects and S. R. Bourne with the shell.

REFERENCES

1. D. M. Ritchie and K. Thompson, "The UNIX Time-Sharing System," B.S.T.J., this issue, pp. 1905-1929.
2. D. M. Ritchie, S. C. Johnson, M. E. Lesk, and B. W. Kernighan, "UNIX Time-Sharing System: The C Programming Language," B.S.T.J., this issue, pp. 1991-2019.
3. S. C. Johnson and M. E. Lesk, "UNIX Time-Sharing System: Language Development Tools," B.S.T.J., this issue, pp. 2155-2175.

4. H. Lycklama and D. L. Bayer, "UNIX Time-Sharing System: The MERT Operating System," B.S.T.J., this issue, pp. 2049-2086.
5. S. R. Bourne, "UNIX Time-Sharing System: The UNIX Shell," B.S.T.J., this issue, pp. 1971-1990.
6. B. W. Kernighan, M. E. Lesk, and J. F. Ossanna, "UNIX Time-Sharing System: Document Preparation," B.S.T.J., this issue, pp. 2115-2135.
7. K. Thompson and D. M. Ritchie, *UNIX Programmer's Manual*, Bell Laboratories, May 1975, section form(1).
8. N. J. Kolettis, private communication.