Copyright © 1978 American Telephone and Telegraph Company
THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 57, No. 6, July-August 1978
Printed in U. S. A.

UNIX Time-Sharing System:

The UNIX Operating System as a Base for Applications

By G. W. R. LUDERER, J. F. MARANZANO, and B. A. TAGUE

(Manuscript received March 9, 1978)

The intent of this paper is twofold: first, to comment on the general properties of the UNIX operating system as a tool for software product development and as a basis for such products; and second, to introduce the remaining papers of this issue.*

I. A BRIEF HISTORY

Bell Laboratories has employed minicomputers in laboratory work since they first became available. By the early 1970s, several hundred minicomputers were controlling experiments, supporting machine-aided design, providing remote-job-entry facilities for computation centers, and supplying peripheral support for Electronic Switching Systems laboratories. The availability of the C-language version of the UNIX system in 1973 coincided with the emergence of several new factors related to minicomputers at Bell Laboratories:

(*i*) The cost, reliability, and capacity of minicomputers—especially improvements in their peripherals—made applications possible that were previously not economical.

(*ii*) Minicomputer-based systems were being selected for installation in operating telephone companies to assist in the administration and maintenance of the telephone plant.

* UNIX is a trademark of Bell Laboratories.

(*iii*) Many such projects were started in a period of a few months, which meant that many engineers were suddenly shifted into minicomputer software development.

(*iv*) The pressures to develop and install these systems rapidly were very great.

Needless to say, the same factors applied to the laboratory uses of minicomputers, but not on the same scale. Product planning estimates suggested that over 1500 such minicomputer-based support systems would be installed in the Bell System by the early 1980s.

The developers of each of these early minicomputer-based projects had to either write their own operating system or find a suitable system elsewhere. The UNIX operating system had become available from the Bell Laboratories computing research organization, and projects were encouraged to use it for their software development. Most projects originally planned to use the UNIX system to develop their own special-purpose operating systems for deployment with their applications. However, all projects were encouraged to consider the UNIX system for deployment with their applications as well. These early projects found that the UNIX operating system provided excellent programming and documentation tools, and was significantly better than the vendor-supplied alternatives then available; the C language and the UNIX system provided a means to rapid development, test, and installation of their software products. Most of these projects found that the UNIX system, with some local modifications, could be used not only for program development, but also as the base for the product itself.

No central support of the UNIX system—i.e., counseling, training, bug fixing, etc.—was available or promised to these pioneer projects. Documentation, aside from a good user's reference manual and the C language listings, was also lacking. The first projects were handicapped by having to supply their own UNIX system support. In spite of this added load, the UNIX system still proved a better choice than the vendor-supported alternatives. Central support and improved documentation were subsequently provided in 1974.

Many requests for improvements to the UNIX system came from telephone-related development projects using it as the base for their products. These projects wanted increased real-time control capabilities in the UNIX system, as well as improved reliability. Even though the systems produced by these projects were ancillary to the telephone plant, Bell System operating telephone companies increasingly counted on them for running their business. Reliability of these systems became an important issue, and significant support effort was devoted to improving the detection of hardware failures by the operating system. Mean-time-to-failure was generally adequate for most projects; mean-time-to-repair was the problem.

Improvements were made that allow UNIX processes to communicate and synchronize in real time more easily. However, additional real-time features were needed to control the scheduling and execution of processes. In another research organization, H. Lycklama and D. L. Bayer developed the MERT (Multi Environment Real-Time) operating system. MERT supports the UNIX program-development and text-processing tools, while providing many real-time features. Some projects found MERT to be an attractive alternative to the UNIX system for their products. This led, in 1978, to the support of two versions of UNIX: the time-sharing version (UNIX/TS) and the real-time version (UNIX/RT). UNIX/TS is based on the research version of the UNIX system with additional features found to be useful in a time-sharing environment. UNIX/RT is based on MERT and is tailored to the needs of projects with real-time requirements. The centrally supported UNIX/TS will become the basis for PWB/UNIX as described by Dolotta et al., and will provide computation-center UNIX service. Real-time projects are currently evaluating UNIX/RT, and real-time features for UNIX are still a matter for continuing investigation and study.

The UNIX and MERT operating systems have found wide acceptance both within the Bell System and without. There are currently 48 Bell Laboratories projects using the UNIX system and 18 using MERT. Some 24 of these projects are developing products for use by the Bell System operating telephone companies and have already installed over 300 UNIX system-based products, with the installation rate still accelerating. In addition, another 10 projects are using PWB/UNIX, and many operating companies are evaluating PWB/UNIX for programming and text-processing tasks within their companies. Outside the Bell System, over 300 universities and commercial institutions are using the UNIX operating system. An important by-product of university use is the growing availability of trained UNIX programmers among computer science graduates.

II. WHY UNIX AND C?

Each of the following papers suggests why the UNIX system was selected for a particular purpose, but some common themes are worth emphasizing. As was suggested above, development projects initially selected the UNIX system as a program-development environment. The earlier paper by Dolotta et al. admirably summarizes the general case for using the UNIX system. Minicomputer-based projects, however, also have incorporated the UNIX system in their final product, and this usage raised new requirements and concerns. We shall attempt to categorize briefly these new requirements and explore their implications.

First, and most obvious, the products of many projects could be considered simply a specialized use of a dedicated time-sharing service; because the UNIX system appeared to be the most efficient available minicomputer time-sharing system, it was an obvious choice for such projects. Its flexibility, generality, and multiprogramming efficiency were just as advantageous for the applications programs as for program developers. Even the existing scheduling and priority mechanisms were adequate for some applications.

Because it was designed for the programming of the UNIX system itself, the C language could also be used to implement even the most complex subsystems. While almost all applications must communicate with special devices unique to their projects, C has an elegant mechanism for solving this problem without special language extensions. It depends upon the fact that the PDP-11 architecture presents all device data and control registers as part of the operating system's virtual address space. These registers can be accessed by assigning absolute constants to pointer variables defined and manipulated by standard C programs.

III. EXTENSIONS FOR REAL-TIME APPLICATIONS

Most projects with real-time applications found the UNIX system wanting in several areas:

(*i*) Interprocess communication.
(*ii*) Efficient handling of large files with known structure.
(*iii*) Communication with special terminals, or using special line protocols.
(*iv*) Priority scheduling of real-time processes.

The general theme is time efficiency; the standard UNIX system already provides all the required functions in some form. What was needed was the ability to "tune" and control these functions in the context of each application. The papers that follow provide three different answers to this problem:

(*i*) Modify and extend the standard UNIX system to provide the additional function or efficiency required.
(*ii*) Adopt MERT, which is a version of the UNIX system restructured to include many real-time functions.
(*iii*) Distribute the application between a central standard UNIX operating system and coupled microprocessors or minicomputers that handle the real-time activities.

Design of real-time applications is a very difficult area, and we expect to see continuing controversy over the best way to handle such applications. The paper by Cohen and Kaufeld argues eloquently why extending the UNIX system was the proper answer for the Network Operations Control System project, while the paper by Nagelberg and Pilla is equally persuasive in arguing that adoption of MERT was the right answer for the Record Base Coordination System project. The papers by Wonsiewicz et al. and Rovegno both describe successful delegation of the real-time aspects of the application to microprocessors connected to the standard UNIX configuration.

The project described by Wonsiewicz et al. is especially relevant. The designers originally selected MERT for their central machine, but in the end they did not use its real-time features. They were able to put all time-sensitive processing into LSI-11 microprocessors in the individual laboratories, and then switched from the MERT system to the UNIX system on their central machine to gain the greater robustness and reliability of UNIX (the MERT system is newer than the UNIX system, more complex, and less well-shaken-down by users).

IV. RELIABILITY AND PORTABILITY

Anything written on minicomputer applications in the telephone system would be remiss if it did not mention the issues of reliability and software portability. The introduction of commercial minicomputer hardware with its 5-year support horizons into the telephone plant that has a 40-year-support tradition raises some obvious questions in the area of reliability and field support. The Bell System investment in applications software must be preserved over long periods of time in the face of rapid evolution of the underlying hardware technology and economics.

The basic reliability of the UNIX software is very good. Note, for example, the comments in Section 8.1 of the paper by Dolotta et al. Using an operating system other than that supplied by the vendor complicates hardware maintenance by the vendor. Effort has been expended in making the UNIX system report errors to the vendor's field engineers in the language and formats that are used by their own diagnostic software. The UNIX system is being improved in its ability to report its own hardware difficulties, but this cannot be carried very far without redesigning the hardware. None of the current UNIX-based systems in the Bell System operating telephone companies directly handle customer traffic, which significantly reduces the reliability requirements of these systems. To achieve the reliability required of Bell System electronic switching offices would necessitate a large and carefully coordinated hardware and software effort.

One of the goals in using the UNIX operating system in telephone applications is to insulate the applications code as much as possible from hardware variations that are driven by the vendors' marketing goals and convenience. By controlling the UNIX system, applications code can be largely protected from the vagaries of the underlying hardware. Such hardware independence is clearly a matter of degree, although full portability of the UNIX system across several different hardware architectures without affecting applications is the ultimate goal. Currently, the UNIX applications code moves quite easily across the PDP-11 line. Experiments are under way to move UNIX to different vendors' hardware, as described by Johnson and Ritchie earlier in this issue. It is already clear that projects must exercise care in how applications are written if they are to move easily from one vendor's architecture to another. Fortunately, much of the needed "care" is simply good C coding style, but unfortunately, precise, complete rules that guarantee portability are proving both complex and a bit slippery. However, the basic idea of using UNIX as an insulating layer continues to be the most attractive option for preserving minicomputer applications software in the face of hardware changes.

V. THE UNIX APPLICATION PAPERS

The six papers that follow describe a spectrum of applications built upon the UNIX and MERT operating systems. The first paper, by Wonsiewicz et al., describes a system handling the automation of a number of diverse instruments in a materials-science research laboratory. In the second paper, Fraser describes a UNIX system used to build an engineering design aid for fast development of customized electronic apparatus. The user works at an interactive graphics terminal with a data base of standard integrated circuit packages, generating output ready to drive an automatic wiring machine. The system makes judicious use of special hardware and software facilities available in only two of the large Bell Laboratories computation centers, which are accessed via communication links. In the application described in the third paper, by Rovegno, the UNIX system is used both as a tool and as a model for the development of microprocessor support software. Whereas a UNIX system was initially used to generate and load microprocessor software in a time-sharing mode, many of its features were then carried over into a small, dedicated microprocessor-based support system. The fourth paper, by Pekarich, shows the use of a UNIX system in a development laboratory for electronic switching systems. It replaces the control portion of a large switching machine, illustrating the ease of interfacing to several specialized devices in the telephone plant.

Whereas these four papers deal with one-of-a-kind systems in research or development environments, the last two papers describe the UNIX system-based products that are replicated throughout the Bell System. The paper by Nagelberg and Pilla describes the MERT-based Record Base Coordination System, which coordinates the activities of several diverse data base systems. Ease of change, even in the field, is the overriding requirement here; this pertains to the interfaces as well as to the algorithms, which are all implemented in the UNIX command language (shell). The paper by Cohen and Kaufeld deals with the Network Operations Control System. It represents the top level of a hierarchy of systems that collect telephone traffic data and control the switching network. Characterized by well-defined and stable interfaces and stringent performance requirements, the design of this system exemplifies how real-time requirements can be met by modifying the UNIX operating system.

VI. SUMMARY

The UNIX system has proven to be an effective production environment for software development projects. It has proven to be an appropriate base for dedicated products as well, though it has often required modification and extension to be fully effective. The near future promises better real-time facilities and some significant portability advantages for the UNIX development community.