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

UNIX Time-Sharing System:

Foreword

by M. D. McILROY, E. N. PINSON, and B. A. TAGUE
(Manuscript received March 17, 1978)

Intelligence ... is the faculty of making artificial objects, especially tools to make tools. — Bergson

UNIX is a trademark for a family of computer operating systems developed at Bell Laboratories. Over 300 of these systems, which run on small to large minicomputers, are used in the Bell System for program development, for support of telephone operations, for text processing, and for general-purpose computing; even more have been licensed to outside users. The papers in this issue describe highlights of the UNIX family, some important uses, and some UNIX software tools. They also attempt to convey a feeling for the particular style or outlook on program design that is both manifest in UNIX software and promoted by it.

The UNIX story begins with Ken Thompson's work on a cast-off PDP-7 minicomputer in 1969. He and the others who soon joined him had one overriding objective: to create a computing environment where they themselves could comfortably and effectively pursue their own work—programming research. The result is an operating system of unusual simplicity, generality, and, above all, intelligibility. A distinctive software style has grown upon this base. UNIX software works smoothly together; elaborate computing tasks are typically composed from loosely coupled small parts, often software tools taken off the shelf.

The growth and flowering of UNIX as a highly effective and reliable

time-sharing system are detailed in the prizewinning ACM paper by Ritchie and Thompson that has been updated for this volume. That paper describes the operating system proper and lists the important utility programs that have been adopted by descendant systems as well. There is no more concise summary of the UNIX time-sharing system than the oft-quoted passage from Ritchie and Thompson:

> It offers a number of features seldom found even in larger operating systems, including
> (*i*) A hierarchical file system incorporating demountable volumes,
> (*ii*) Compatible file, device, and inter-process I/O,
> (*iii*) The ability to initiate asynchronous processes,
> (*iv*) System command language selectable on a per-user basis,
> (*v*) Over 100 subsystems including a dozen languages.

Implementation details are covered in a separate paper by Thompson. Matters of efficiency and design philosophy are considered in a retrospective paper by Ritchie.

The most visible system interface is the "shell," or command language interpreter, through which other programs are called into execution singly or in combination. The shell, described by Bourne, is actually a very high level programming language that talks about programs and files. Particularly noteworthy are its notations for input-output connections. By making it easy to combine programs, the shell fosters small, coherent software modules.

The UNIX system and most software that runs under it are programmed in the general-purpose procedural language C. C provides almost the full capability of popular instruction sets in a setting of structured code, structured data, and modular compilation. C is easy to write and (when well-written) easy to read. The language and the philosophy behind it are covered by Ritchie, Johnson, Lesk, and Kernighan.

Until mid-1977, the UNIX operating system and its variants ran only on computers of the Digital Equipment Corporation PDP-11 family. In an interesting exercise in portability, Johnson and Ritchie exploited the machine-independence of C to move the operating system and the bulk of its software to a quite different Interdata machine. Careful parameterization and some repackaging have made it possible to use largely identical source code for both machines.

Variations

Three papers by Bayer, Lycklama, and Christensen describe

variations on the UNIX operating system that were developed to accommodate real-time processing, microprocessor systems, and laboratory support applications. They were motivated by the desire to retain the benefits of the UNtX system for program development while offering different trade-offs to the user in real-time response, hardware requirements, and resource management for production programs. Many UNIX utilities—especially those useful for writing programs and processing text—will run under any of these variant systems without change.

The MERT operating system (Lycklama and Bayer) provides a generalized kernel that permits extensive interprocess communication and direct user control of peripherals, scheduling, and storage management. Applications with stringent requirements for real-time response, and even different operating systems (in particular, UNtX) can be operated simultaneously under the MERT kernel.

The microprocessor version of the UNIX operating system (Lycklama) and the Satellite Processing System that shares process execution between one big and one tiny machine (Lycklama and Christensen) involve other trade-offs between efficiency and resource requirements. Both also may be looked upon as vehicles for applications in which one wishes to delegate some sticky part of the job—frequently involving real-time demands—to a dedicated machine. The application described later in the issue by Wonsiewicz, Storm, and Sieber is a particularly interesting example involving UNtX, the microprocessor system, and the Satellite Processing System.

Software Tools

Perhaps the most widely used UNtX programs are the utilities for the editing, transformation, analysis, and publication of text of all sorts. Indeed, the text-processing utilities covered by Kernighan, Lesk, and Ossanna were used to produce this issue of the B.S.T.J. Some more unusual applications that become possible where text processors and plenty of text are ready at hand are described by McMahon, Morris, and Cherry.

UNIX utilities are usually thought of as tools—sharply honed programs that help with generic data processing tasks. Tools were often invented to help with the development of UNIX programs and were continually improved by much trial, error, discussion, and redesign, as was the operating system itself. Tools may be used in combination to perform or construct specific applications.

Sophisticated tools to make tools have evolved. The basic typesetting programs nroff and troff covered by Kernighan, Lesk, and Ossanna help experts define the layouts for classes of documents; the resulting packages exhibit only what is needed for one particular type of document and are easy for nonspecialists to use. Johnson and Lesk describe Yacc and Lex, tools based in formal language theory that systematize the construction of compiler "front ends." Language processors built with the aid of these tools are typically more precisely defined and freer from error than hand-built counterparts.

The UNIX system was originally designed to help build research software. What worked well in a programming laboratory also worked well on modest projects to develop minicomputer-based systems in support of telephone company operations. Such projects are treated in the final group of papers and are more fully introduced by Luderer, Maranzano, and Tague. The strengths of this environment proved equally attractive to large programming projects building applications for large computers with operating systems that were less tractable for program development. The PWB/UNIX extensions discussed by Dolotta, Haight, and Mashey provide such projects with a "front end" for comfortable and effective program development and documentation, together with administrative tools to handle massive projects.

Style

A number of maxims have gained currency among the builders and users of the UNIX system to explain and promote its characteristic style:

(*i*) Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new "features."

(*ii*) Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don't insist on interactive input.

(*iii*) Design and build software, even operating systems, to be tried early, ideally within weeks. Don't hesitate to throw away the clumsy parts and rebuild them.

(*iv*) Use tools in preference to unskilled help to lighten a

programming task, even if you have to detour to build the tools and expect to throw some of them out after you've finished using them.

Illustrations of these maxims are legion:

(*i*) Surprising to outsiders is the fact that UNIX compilers produce no listings: printing can be done better and more flexibly by a separate program.

(*ii*) Unexpected uses of files abound: programs may be compiled to be run and also typeset to be published in a book from the same text without human intervention; text intended for publication serves as grist for statistical studies of English to help in data compression or cryptography; mailing lists turn into maps. The prevalence of free-format text, even in "data" files, makes the text-processing utilities useful for many strictly data processing functions such as shuffling fields, counting, or collating.

(*iii*) The UNIX system and the C language themselves evolved by deliberate steps from early working models that had at most a few man-months invested in them. Both have been fully recoded several times by the same people who designed them, with as much mechanical aid as possible.

(*iv*) The use of tools instead of labor is nicely illustrated by typesetting. When a paper needs a new layout for some reason, the typographic conventions for paragraphs, subheadings, etc. are entered in one place, then the paper is run off in the new shape without retyping a single word.

To many, the UNIX systems embody Schumacher's dictum, "Small is beautiful." On the other hand it has been argued by Brooks in *The Mythical Man Month,* for example, that small is unreal; the working of a handful of people doesn't extrapolate to the world of big jobs. We agree only in part, for the present volume demonstrates with unusual force another important factor: intelligently applied computing technology can compress jobs that used to be big to manageable size. The first system had only about 5 man-years' work in it (including operating system, assembler, Fortran, and many other utilities) when it began to be used for Bell System projects. It was, to be sure, a taut package that lacked the gamut of libraries, languages, and support for peripheral equipment typical of a large commercial system. But the base was unusually

pliable and responsive; new facilities usually could be added with much less work than is required by corresponding features in other systems.

The UNIX operating system, the C programming language, and the many tools and techniques developed in this environment are finding extensive use within the Bell System and at universities, government laboratories, and other commercial installations. The style of computing encouraged by this environment is influencing a new generation of programmers and system designers. This, perhaps, is the most exciting part of the UNIX story, for the increased productivity fostered by a friendly environment and quality tools is essential to meet ever-increasing demands for software. UNIX is not the end of the road in operating system innovations, but it has been a significant step that Bell Laboratories people are proud to have originated.