The UNIX Operating System and the XENIX Standard Operating Environment

Robert B Greenberg
XENIX Product Manager
Microsoft
10800 NE Eighth, Suite 819
Bellevue WA 98004

Never has there been a greater demand for software that is easy to use and maintain, and independent of the hardware on which it runs. As the price of software rapidly outpaces that of computers, the need to increase software productivity and reduce duplication of effort has become paramount.

Microsoft's XENIX operating system offers one solution to the software crisis developing in the microcomputer world. Unlike the operating systems offered for 8-bit machines, the XENIX system is a powerful multiuser timesharing system with hundreds of utilities and is the basis for a highly productive software development environment and a general-purpose applications system.

The XENIX operating environment combines two key elements: the design of the widely acclaimed UNIX operating system and the inclusion of the major high-level languages that are standard within the 8-bit microcomputer world (see figure 1). Microsoft's transport of the XENIX system to major 16-bit microprocessors has made it the first hardware-independent operating system.

The heart of the XENIX system is the UNIX operating system developed at Bell Laboratories and licensed by Western Electric. The UNIX system's elegant design combines power, flexibility, and simplicity, and its vast array of software utilities greatly increases productivity. Thus, the UNIX system is an ideal candidate to serve as a solution to the software crisis.

Microsoft plans to make the XENIX operating system (which is an enhanced version of the UNIX system) into a commercial standard. And, in addition to supporting and enhancing the operating system proper, Microsoft will adapt high-level languages, such as its BASIC interpreter and compiler, FORTRAN, Pascal, and COBOL, and other software tools, such as data-base management and communications software, to run under the XENIX operating system.

The XENIX system is one approach to solving the software crisis developing in the microcomputer world.

To understand the elegance of the basic UNIX design and the further enhancements in the XENIX system, we must take a closer look at the software. In this article, I will describe the main features in the UNIX operating system, discuss some of its strengths and weaknesses, and conclude with a discussion of the evolution of the XENIX operating environment from the UNIX operating system, and how it can help solve critical software issues. First, a historical overview.

UNIX is a trademark of Bell Laboratories.
XENIX is a trademark of Microsoft.

Origins of the UNIX OS

The UNIX operating system was originally developed at Bell Laboratories by Ken Thompson, an employee engaged in various programming research projects. With access to an abandoned DEC PDP-7 computer that had no software, Thompson decided in 1969 to write a set of programs that would aid him in software research. Over a period of several years, and with the help of fellow researcher Dennis Ritchie, this set of programs evolved into a full operating system. By 1972, it was recoded for the DEC PDP-11 computer in a newly designed high-level language, called C. The system gained recognition within the Labs and their parent company, Western Electric.

Word of the quality of Thompson and Ritchie's UNIX operating system spread rapidly. Universities, in particular, expressed interest in obtaining UNIX, and in 1973, Western Electric agreed to distribute the system to nonprofit organizations and promptly licensed several dozen educational institutions, including Columbia University, the University of Alberta (Canada), The Children's Museum (Boston), Princeton University, and Harvard University. By 1975, UNIX had become sufficiently popular in the academic world to justify the

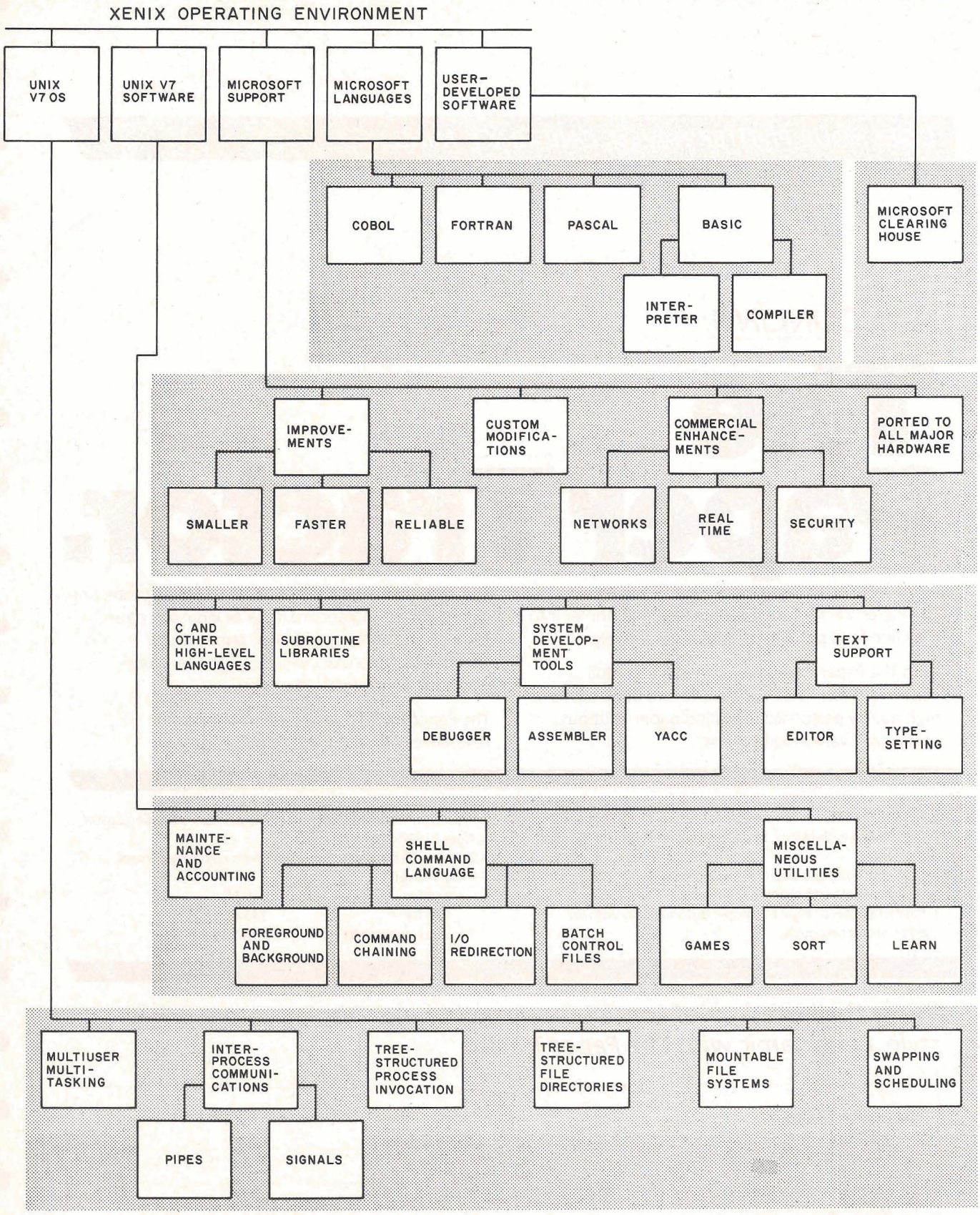


Figure 1: *Microsoft's XENIX operating system. The five "layers" of the XENIX software structure are shown. XENIX, a superset of Bell Laboratories' UNIX operating system developed in the early 1970s, has a hierarchical structure. Each of the five layers depends on the layers beneath it for its operation. The bottom two layers represent the latest version of UNIX (version 7). The remaining three layers are the refinements that combine to make the XENIX system.*

creation of a UNIX users' organization, later called USENIX.

The first public release of the UNIX operating system, labeled version 5, was an unpolished snapshot of a research project that was still evolving. It was replaced in 1975 with version 6, a system that is still operating today at many sites. UNIX continued to evolve, benefitting from the feedback it received from scores of internal and external test sites.

In January 1979, Western Electric released version 7. By this time, hundreds of man-years' effort has been expended on UNIX's design and software utilities, with most of the system coded in C. Research had proven that UNIX was compatible with the concepts of memory-limited computers, machine transportability, networks, and multiple-processor designs.

Unfortunately, there was no single standard design for UNIX. Because the operating system was simple and easy to change, almost every site altered it to meet their specific needs. Harvard, the University of California at Berkeley, and the RAND Corporation each offered a set of modifications. A number of incompatible versions of UNIX existed within Western Electric.

In addition, there has been a legal impediment to the UNIX system's distribution. The system is available essentially free-of-charge for educational institutions. Legally, however, Western Electric cannot be in the software business, so the commercial world is offered the operating system under noncompetitive terms: source code as is and no warranty, support, or maintenance—a steep fee for software that was never intended to serve commercial applications outside of Western Electric.

It had become clear that the support of a commercial software company was essential if UNIX was to become a software standard. In August of 1980, Microsoft announced that it would offer and support XENIX, a commercial version of the operating system, on 16-bit microprocessors. Working closely with Western Electric and a newly formed commercial users' organization, Microsoft intends to establish a standard industry version of UNIX that can provide a highly productive environment worthy of meeting the challenges of software development in the 1980s.

UNIX Design Goals

Two aspects of UNIX's origin have contributed to its design: (1) it was created in a few man-years by two people, and (2) the implementers were also major users of the system. The result is a polished, consistent, coherent design. UNIX achieves great power and flexibility, including compatible interfacing between all its features, without resorting to a large, complex program. An experienced system programmer can understand the entire operating system in weeks, rather than months.

The UNIX system's design goals unite various features supported by the UNIX sytem into a consistent and simple whole. The first design goal is to support a very basic level of functionality within the operating system itself, relying on normal user programs to provide sophistication. Such features as line printer queuing, login/logout, monitor commands, and file access methods are implemented as normal user programs instead of operating-system functions. This approach, which reduces the overall complexity of the system, has several advantages. Functions are more modular, and therefore easier to debug, features can be altered and upgraded without stopping the operating system, and alterations made to one feature are less likely to affect the rest of the system. Finally, individual users may create personal versions of certain features.

The second design goal is generality—that is, having a single method serve a variety of related purposes. For example, the same system calls are used to read and write disk files, devices, and interprocess message buffers. Likewise, the same naming, aliasing, and access protection mechanisms apply to data files, directories, and devices. As a final example, the same mechanism is used to trap software interrupts, user abort requests, and processor traps. The benefits of generality extend well

beyond the simplicity of design; UNIX programming style is notably flexible, extensible, easily learned, and easily debugged.

The third goal is to accomplish large tasks by combining several small tasks whenever possible. UNIX's filters are an excellent example. A filter is a program that processes a single stream of input to generate one output stream. The UNIX system has a large variety of filters, including those that perform multicolumn formatting, string replacement, text processing, character translation, sorting, and graphics interfacing. Programs that generate output, such as the assembler, do not include facilities for listings; this task is accomplished by feeding programs directly to the various filters. This keeps the large programs simple to use, lets a user learn about each filter separately, and allows for special combinations of formatting without multiplying the options that each program would then have to support. It also leads to a uniform appearance of formatted output and the commands needed to produce it, and yields all the benefits of modular solutions to complex problems.

The vast number of utilities provided with the system and the ease of linking them together via pipes provide a surprising amount of functionality. For example, to find out how many people are currently using the system, you need only feed the output of the system "who" command to the utility that prints the number of lines in its input. Thus, the command line:

```
who | wc -l
```

causes the output of the who command, which might look like:

```
arw       console    Jan 30 14:20
bobg      tty00      Jan 30 01:00
henry     tty01      Jan 30 12:50
gordon    tty03      Jan 29 10:08
```

to be fed to the program "wc," for "word count." The -l option tells wc, which normally prints the number of characters, words, and lines in a file, that we only want to see the number of lines. Thus, this composite command prints a number which is the number of users on the system:

```
>who | wc -l
4
>
```

As a final step, we can create a file called "users," which contains the line:

```
who | wc -l
```

Typing "users" causes the command interpreter (or shell) to execute that line, and type the number of current users. We have now created a new system command.

A more dramatic example is shown in the following sequence: take a program that puts each text word in a file (or files) onto a separate line. Connect the output to a program that sorts lines into alphabetical order.

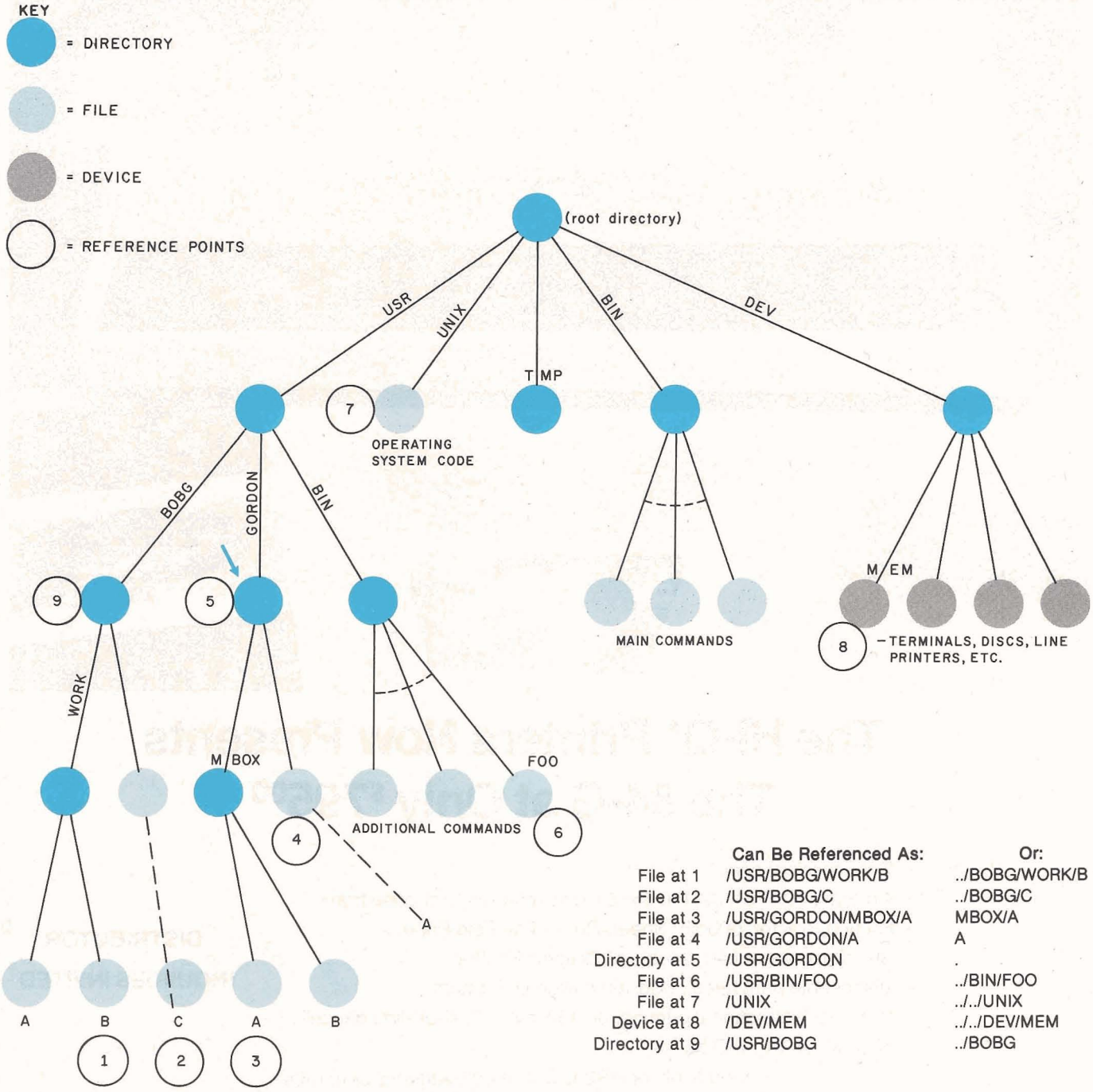
The output is a sorted list of all words in the text file(s). This list is fed to the program "uniq", which removes adjacent duplicate lines. The result is a data stream that contains one line for each different word in the original file(s). This stream is in turn connected to a program that reports differences between two files (one file being a list of 30,000 words from the dictionary). Thus, typing the line:

```
prep file | sort | uniq | comm wdlist
```

will result in a list of words present in "file" but not present in "wdlist". Without writing a line of code, you have created a simple spelling program! Now, by creating a file called "spell", which contains the line:

```
prep $* | sort | uniq | comm
/usr/dict/words
```

you have created the command "spell". Note that the "$*" is replaced by the command line interpreter with the arguments typed to the spell command. The UNIX sytem's command



	Can Be Referenced As:	Or:
File at 1	/USR/BOBG/WORK/B	../BOBG/WORK/B
File at 2	/USR/BOBG/C	../BOBG/C
File at 3	/USR/GORDON/MBOX/A	MBOX/A
File at 4	/USR/GORDON/A	A
Directory at 5	/USR/GORDON	.
File at 6	/USR/BIN/FOO	../BIN/FOO
File at 7	/UNIX	../../UNIX
Device at 8	/DEV/MEM	../../DEV/MEM
Directory at 9	/USR/BOBG	../BOBG

Figure 2: *Hierarchical structure of the names and conventions for getting to any reference point in a typical XENIX file structure. In this example, it is assumed that the user is at reference point 5 (blue arrow). A list of instructions for getting to the various reference points appears beneath the diagram. (The file and directory labels shown here are actual labels used in the author's system.) To get to file 1, the user types "/USER/BOBG/WORK/B". XENIX then progresses down the tree from the root directory (at top) to the branches USR, BOBG, WORK, and B, arriving at point 1. Alternatively, the user can use the command "../BOBG/WORK/B", where ".." refers to the parent node of the node currently in use. In XENIX, "." refers to the node itself.*

interpreter, the shell, is a fully interactive language in its own right.

UNIX Operating System Design

The UNIX design introduces few new concepts because it borrows heavily from the better aspects of previously existing systems. UNIX contains numerous features found in the MULTICS and AOS operating sytems, and the language C is modeled after BCPL. However, the coherence and simplicity with which the chosen features interact result in an unusually elegant design that has great merit of its own.

The UNIX operating system supports a multiuser, multitasking environment. Each user has full access to the resources of the computer on a timesharing basis. UNIX implements scheduling and swapping algorithms that allow the processor and memory to service more tasks, seemingly simultaneously, than would otherwise be possible. UNIX also includes various protection schemes that protect each user from the others. This functionality contrasts markedly with the current microcomputer systems that simplify hardware operation by providing device drivers but make little attempt to extend the computer's utility.

The UNIX file system is a recursive structure originating from a *root directory*. The root directory contains the names of files and subdirectories; the subdirectories contain names of other files and additional subdirectories, etc. When a user logs into the system, he is assigned a specific subdirectory as his current working directory. Full path names for files consist of a possibly null sequence of subdirectories separated by a slash, beginning with either the root or the current working directory, and followed by the file name. By convention, the file in each subdirectory called ".." refers to the parent directory (see figure 2). Thus the user has a concept of local and global files neatly organized into directory groupings.

File names refer to data files, the directories themselves, character devices such as user terminals, block devices such as magnetic tape, file systems mounted onto other disk devices, and interprocess communications devices known as *multiplexed pipes*. Multiple names (called aliases) can be assigned to any of these objects. A set of information, including owner and access permissions, is stored with each object; the directory entries only specify names for the objects.

Programs communicate with their environment with read and write calls directed to a set of open files. Each program starts with three open files: *standard input*, *standard output*, and *error output*. Normally, these files are connected to the user's terminal, but a powerful command-language program, the shell, allows easy and invisible reassignment of these channels. A program can also open any other object (file, device, etc) named in the file system to which it has appropriate access permission. Using a special call, a program can create pipes, data channels that allow for communication between the program and any other programs connected to an end of the pipe.

All I/O (input/output) operations are performed as byte streams, with all channels appearing to contain a sequence of bytes until a globally defined end-of-file condition is indicated. Random access is also supported, using a call to reposition within the stream. Neither record sizes nor file types are imposed by the operating system. The system handles all interrupts and buffering, and each I/O call is suspended until the requested I/O operation can be completed. All devices, files, and pipes are treated identically (with minor exceptions), which greatly simplifies I/O routines.

A program may initiate another program by issuing a system call to duplicate itself. The two programs then operate independently, with

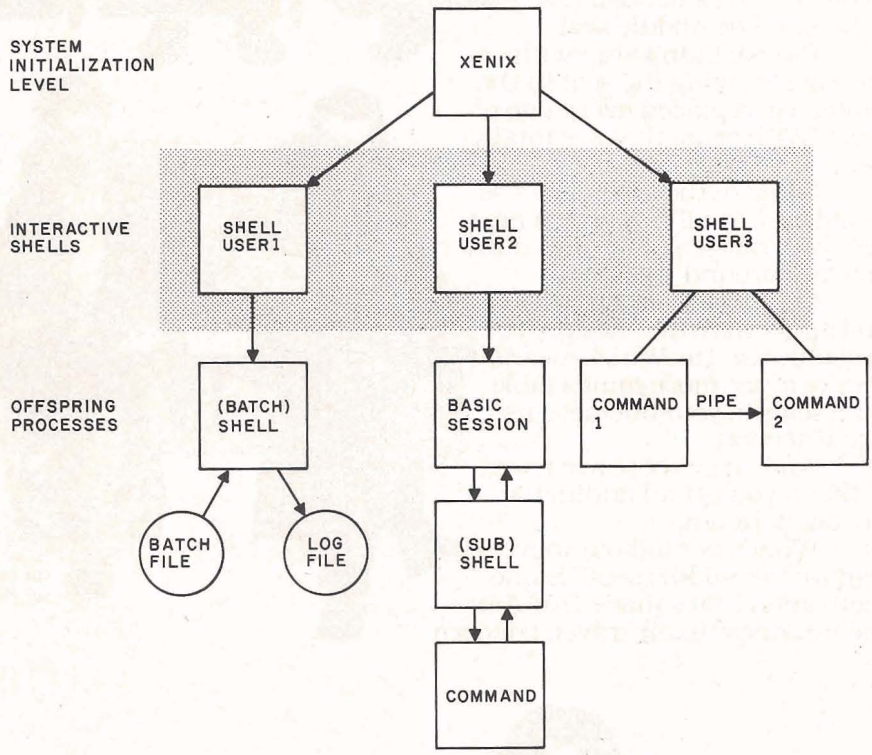


Figure 3: *Tree-structured process hierarchies in the XENIX system. Three users are currently on line. The term "shell" refers to that portion of the XENIX operating system program that "surrounds" the operating system and allows it to communicate with the outside world. User 1 is running a batch shell that is executing commands from a file. User 2 has suspended a BASIC session and entered a subshell to issue a command at the system-monitor level, perhaps to send a message to another user. User 2 can then return to BASIC and resume the session. User 3 has executed a command whose output is piped through a second command.*

UNIX timesharing between them (see figure 3). Typically, the parent process waits for the completion of its child, and the child process executes another program in the file system by issuing a system call. However, both programs may continue execution in parallel. To synchronize their operation, they can communicate via the file system, pipes, or signals. Signals are software asynchronous interrupts that are issued by one program to another to cause the second program to interrupt its execution, process the signal, and then resume normal execution. Signals are also generated by user interrupt requests and software failures, such as divide-by-zero.

Thus, when a user compiles and links a program test.c by typing:

```
>cc test.c
```

the shell runs the C compiler (cc) as a child process. After it has spawned the child process, the shell puts itself to sleep. When the child process (the C compiler) finishes, the shell awakens and issues another prompt. However, by simply adding an ampersand character to the command line:

```
>cc test.c &
```

you can instruct the shell not to sleep, but rather to return immediately for another command. You can then edit your documenation or some further program, while the first one is compiling. Note that typing:

```
>filename
```

causes the shell to run a copy of itself as a child. This child shell then executes, one by one, the commands in "filename." By simply adding the "&" character to the following line:

```
>filename &
```

you now have the capabilities of a full batch system, for free, as a result of the UNIX system's flexibility.

This section has presented a brief overview of the UNIX system features. A more complete description is available in documents from Microsoft, Western Electric, and a number of universities. I will conclude this section with a discussion of an excellent example of UNIX's multitasking abilities.

Multitasking

The multitasking and interprocess communication features of the UNIX system provide power that is unavailable in existing 8-bit computer systems. RITA, a large interpreter language for UNIX that I helped create for the RAND Corporation, provides an extensive example of the utility of these features. The RITA interpreter consists of over 100 K bytes of instructions and more than 64 K bytes of data—much larger than the current limit on UNIX program size. The solution was to split RITA into three separate programs that communicate though the use of five pipes, as illustrated in figure 4. Furthermore, separate programs are created by the interpreter to edit programs, read RITA news files, and perform UNIX commands, such as obtaining

access to networks. Several files are written for analysis by still other programs. All this multitasking takes place invisibly: the user still thinks he or she is running a single program.

A further benefit of multitasking and device-independent I/O is an unexpected feature of RITA's three-program arrangement. Normally, the first program, UFE (user front end) allows you to type and edit program statements, which are then converted to internal form by the second program, the parser, which in turn stores them in the third program, the monitor, for evaluation. The UFE also allows the statements to be entered from a disk file; however, due to the complex parser program, loading a large file is too time consuming for many applications. A slight alteration to the UFE, the program which creates the other two programs and the five pipes, provides the solution. The new UFE (now called RC for RITA compiler), which requires no changes to the parser or monitor, funnels the output of the parser, normally fed to a pipe, into a disk file. Thus, RC produces "compiled" files whose contents can be fed directly into the monitor, bypassing the parser, when later loaded by RITA's UFE.

An Assessment of UNIX

UNIX offers unparalleled power for such a straightforward system. For the programmer, the system is easy to learn and offers immediate functionality, even for beginners. For more experienced users, the wealth of software tools leads to a more productive environment than less complete systems.

In addition, the UNIX operating system comes with hundreds of utilities and software tools that make it a complete software development environment. There is software for accounting, text editing, formatting and typesetting, high-level languages,

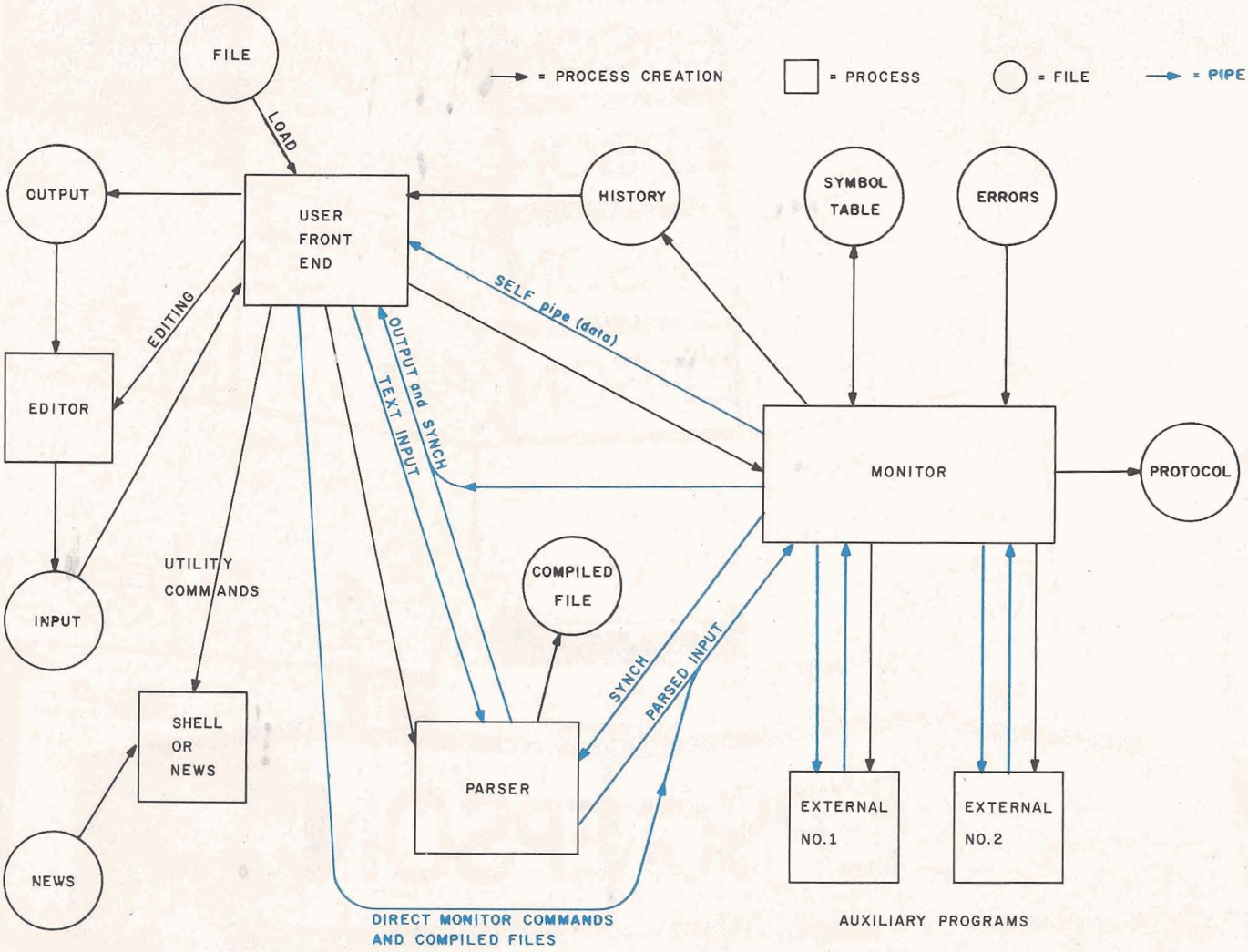


Figure 4: *RITA, a program designed in part by the author to illustrate the multitasking and interprocess communication features of the UNIX system. The RITA interpreter consists of over 100 K bytes of instructions and more than 64 K bytes of data: much larger than the current limits on UNIX program size. The solution to the problem is to split RITA into three separate programs that communicate through the use of five "pipes." A different UFE (user front end) program, called the RITA compiler, can refunnel the output of the parser, normally fed to the monitor, into a disk file. Thus, the RITA compiler produces "compiled" files whose contents can be fed directly into the monitor, bypassing the parser, when later loaded by RITA's user front end. This approach allows the user to load large files that might otherwise require too much time.*

assembly support utilities, sorters and index generators, communication facilities, tools that create parsers and lexical analyzers, graphics, games, mathematical function libraries, maintenance and performance utilities, and a host of file manipulators. Few needs cannot be met through a combination of these existing utilities.

The flexibility of UNIX allows easy alteration of its user interface. Various installations have demonstrated how easy it is to completely alter the appearance of UNIX in order to serve a different class of users. That UNIX cannot be everything to everyone is overshadowed by the fact that, as it is truly general-purpose, it can perform in almost any environment.

UNIX, as supplied by Western Electric, is not without its weaknesses. The general-purpose timesharing design limits UNIX's efficiency in real-time applications, such as process control. Its standard interface is highly terse, and though this is often considered desirable by programmers, the untamed UNIX will frighten almost everyone else. The origins of many of the command names are obscure; examples include a tape command "r" to write to a tape, command "cat" which types files, and "awk", a program for finding patterns in files. However, command names can be easily changed by the user.

UNIX has not been adapted for commercial use, where the issues of reliability, stability during hardware errors, full per-user accounting, reconfigurability for a large variety of environments, and security take on special importance. For example, less expensive disk packs for larger disk drives usually contain bad spots, and UNIX does not automatically adjust for them. In the environment for which the UNIX system was developed, it was cheaper to buy perfect packs than to write a " bad spot avoidance" routine. These issues must be addressed before UNIX can be considered a sturdy, robust, and commercial piece of software.

A crucial problem, and one not restricted to UNIX, is the lack of true applications software. Currently, there are few good accounts payable, invoicing, mailing list, income tax, or data-base management packages. UNIX provides an excellent software production environment because of its wealth of software tools utilities, but the system does not contain a similar variety of application-oriented software.

The XENIX System

Microsoft's XENIX operating system represents an attempt to preserve the strengths of the UNIX design and also meet the needs of the commercial microprocessor industry. To achieve this goal, Microsoft used the system as it was distributed by Western Electric and then added modifications, customizations, improvements, enhancements, support, and additional software.

Modifications included those necessary to transport the UNIX system from the larger PDP-11 minicomputer to the 16-bit microprocessors. Currently scheduled machines include the DEC LSI-11/23, Zilog's Z8001 and Z8002, Intel's 8086 and 286, and Motorola's MC68000. Numerous other processors are also being considered, and Microsoft will then customize the XENIX systems to the specific hardware environments of the various computer systems built around these processors. The company is also working closely with a number of hardware manufacturers to design products that will be capable of efficiently executing the XENIX software.

Improvements will include elimination of known bugs and recoding of certain routines to produce a smaller and faster operating system. XENIX will also incorporate hardware error recovery strategies, automatic file repair after crashes, power-fail and parity-error detection, and similar features, depending on the particular hardware requirements of each XENIX system.

The planned enhancements will add a number of new features to XENIX. These features include record locking, shared data segments, synchronous writing, and improved interprocess communication—all of which are designed to make XENIX commercially viable and more compatible with the newer hardware technologies that involve distributed data processing, networking, and multiple-CPU approaches.

XENIX is a dynamic, evolving system. In its first release, its code was very close to the original UNIX version 7 source. The improvements and enhancements that I have mentioned are part of an evolving process, and the exact selection and specification of features will be developed throughout the course of 1981. Updates to XENIX will result in systems upwardly compatible from its first release.

The adaptation of Microsoft's full line of system software products to XENIX will further strengthen XENIX's role as a software standard. These products, including the BASIC interpreter and compiler, COBOL, FORTRAN, and Pascal, have already established themselves as standards within the 8-bit market; they are also compatible with corresponding ANSI (American National Standards Institute) standards. Standard high-level languages will allow the rapid introduction of existing application software into the XENIX environment.

The XENIX system will offer an ever-expanding variety of software, including data-base management, financial planning, communication, and networking packages. Microsoft is establishing a clearinghouse, wherein quality software running under XENIX may receive widespread distribution, thereby reducing duplication of effort. The combination of the UNIX operating system's strengths and Microsoft's awareness of the needs of the commercial marketplace promises to make XENIX a very powerful defense against the looming software crisis. By establishing a universal operating environment, complete with software tools to increase productivity, flexible design to widen applicability, and multiple microprocessor support to improve availability, Microsoft hopes that XENIX will become the preferred choice for software production and exchange. ■