A UNIX System Implementation for System/370*

W. A. FELTON, G. L. MILLER, and J. M. MILNER

ABSTRACT

This paper describes an implementation of the UNIX(TM) operating system for IBM System/370 computers. In this implementation an underlying Resident Supervisor, adapted from an existing IBM control program, provides machine control and multiprogramming; while a UNIX System Supervisor, adapted from the standard UNIX system kernel, provides the UNIX system environment. This implementation supports multiprocessing, paging, and large-process, virtual address spaces. Terminal handling is done through an outboard terminal processor. This paper describes the software structure, with emphasis on unique aspects of this implementation: multiprocessing and process synchronization, process creation, and outboard terminal handling. Capacity and performance of the UNIX system on large mainframes is also discussed. The first and principle user of the UNIX system for System/370 is the development project for the 5ESS(TM) switching system. This paper also discusses the use of a large mainframe UNIX system for this development. Included in this discussion are the reasons for selecting this system for development, applications software porting, and general experience with mainframe UNIX systems.

I. INTRODUCTION

One of the great strengths of the UNIX operating system is its portability. UNIX system implementations have been done for a variety of computers with greatly varying architectures [1]. Perhaps nowhere is this portability better illustrated than in its implementation for System/370 machines.

Since its introduction by IBM in 1970, System/370 [2] has become the dominant architecture for large computer systems; currently about 70 percent of the large mainframes in the United States follow System/370 architecture. IBM builds a variety of System/370 machines, from relatively small "superminis" to their largest processors. In addition, other manufacturers, such as Amdahl Corporation, build machines that conform to System/370 specifications and can thus run System/370 operating systems and applications. The principal operating system currently used on these machines is IBM’s MVS (Multiple Virtual System), although other operating systems--IBM’s VM/370 and TSS/370, and the University of Michigan’s MTS--are also available.

The idea of a UNIX system implementation on System/370 machines, which would bring the power of these large processors to the UNIX system user, has been discussed for some time. In 1978 we began to seriously study the possibility of such an implementation. Our primary objective was to develop a true version of the UNIX operating system that would be suitable for use in a production environment on System/370 machines, making full use of the features and power of these large machines. We wanted to make the System/370 environment appear to the user and applications programmer as similar as possible to the standard UNIX system environment; in the words of one developer, it should "look, feel, and smell like the UNIX system people are familiar with". At the same time, we wanted a system that would provide reliable, cost-effective production service, as, for example, in a computation center environment. Most of the design for implementing the UNIX system for System/370 was done in 1979, and coding was completed in 1980. The first production system, an IBM 3033AP, was installed at the Bell Laboratories facility at Indian Hill in early 1981. Since then several large IBM System/370 mainframes have been made to run the UNIX system at Indian Hill. In addition, there are installations at Holmdel and Denver.

The first user of the UNIX system for System/370, and currently the largest user, is the development

Published in, and copyright by, AT&T Bell Laboratories Technical Journal, Vol. 63, No. 8 Part 2, October 1984. Photoreproduction for noncommercial use is permitted without payment of royalty provided that each reproduction is done without alteration and that the Journal reference and copyright notice are included on the first page.

project for the 5ESS switch. Even as the system was being developed, the needs of the project were quickly reaching beyond the use of minicomputers. The UNIX operating system was selected as the development system to be used by the programmers developing the switching system software. The UNIX system was selected because of the facilities of the Programmer's Workbench [4] software, which provide the developers with editors, source code control, and software generation systems. Initially, development was done on several PDP-11/70 systems. By late 1980 the project was using nine PDP-11/70 systems to provide the programmer development support environment. These computers were linked together using a commercially available high-speed network with drivers written for the UNIX operating system. The fragmentation of the project over nine computers caused significant additional work. The low-level compiled objects that were compiled on the nine computers had to be networked onto one computer for the final linking before generating the final switching program output. The final products had to be distributed back to the other eight computers so that private changes could be linked into the full system for private testing. Also, periodic auditing had to be done to ensure that all computers had the same common data and that the compilers and other tools remained the same on each system. The project was continuing to grow, and adding more minicomputers was not the best solution, because the auditing and networking overhead would increase on all the minicomputer systems.

Several solutions were considered to the problem of the growing number of minicomputers required for the project. The UNIX operating system with the Programmer's Workbench software provided a better development environment than any other operating system available. In addition, the developers were all trained in using this system and all the software tools had been developed. This led to a requirement that the computer systems selected to solve the problem support the UNIX operating system, as well as provide an order of magnitude more computing power in one system than the PDP-11/70 systems that were being used. This requirement ruled out larger minicomputers such as the VAX-11/780 systems, which offers approximately twice the computing power of the PDP-11/70 system. The IBM 3033AP processor met the requirement with approximately 15 times the computing power of a single PDP-11/70 processor. After studying the problem, the project decided to use the UNIX system for System/370, and requested that the porting be completed and a production grade system be made available in mid-1981.

II. SOFTWARE ENVIRONMENT

We initially thought about porting the UNIX operating system directly to System/370 with minimal changes. Unfortunately, there are a number of System/370 characteristics that, in the light of our objectives and resources, made such a direct port unattractive. The Input/Output (I/O) architecture of System/370 is rather complex; in a large configuration, the operating system must deal with a bewildering number of channels, controllers, and devices, many of which may be interconnected through multiple paths. Recovery from hardware errors is both complex and model-dependent. For hardware diagnosis and tracking, customer engineers expect the operating system to provide error logs in a specific format; software to support this logging and reporting would have to be written. The System/370 architecture lends itself to the use of paging for memory management; the UNIX system used swapping. Finally, several models of System/370 machines provide multiprocessing, with two (or more) processors operating with shared memory; the UNIX system did not support multiprocessing.

Since code to support System/370 I/O, paging, error recording and recovery, and multiprocessing already existed in several available operating systems, we investigated the possibility of using an existing operating system, or at least the machine-interface parts of one, as a base to provide these functions for the System/370 implementation. We needed a well-structured system that could provide a clean interface for UNIX system processes. The system would have to provide all the functions needed by UNIX system processes, or at least be extendible to provide these functions with reasonable effort.

Of the available systems, TSS/370 came the closest to meeting our needs and was thus chosen as the base for our UNIX system implementation [5]. The choice of TSS/370 was a controversial one; it is a little known and inadequately documented system. Still, it came the closest to providing the structure and function needed to support UNIX system processes, and it appeared that it could be enhanced to provide any missing functions with reasonable effort. In 1979 we proposed to IBM that they make the necessary modifications to the TSS supervisor to support UNIX system processes, according to our design. IBM agreed to do so under a program license agreement, and the first version of the enhanced TSS was delivered in 1980.

2.1 Software structure

The UNIX system for System/3 70 comprises three classes of programs, running in different software levels. From highest to lowest, these are:

1. User-level programs, including user-written programs and system-provided programs, such as the shell;
2. The UNIX System Supervisor, which incorporates much of the function and C-language code of the standard UNIX system kernel; and
3. The Resident Supervisor, which supports the multiprogramming of UNIX system processes, provides low-level system calls, and manages the physical system configuration.

Each UNIX system process, comprising a user-level program and the UNIX System Supervisor, executes within its own 16-megabyte virtual memory, in the context of its own virtual machine. The Resident Supervisor controls the resources allocated to these virtual machines, including process scheduling, dispatching, and real storage management.

User programs and the UNIX System Supervisor share the same 16-megabyte process space. The UNIX System Supervisor is located in the upper 8 megabytes of this space; user programs are located in the lower 8 megabytes. "Page 0", the lowest 4096 bytes of the process space, is reserved for Program Status Words (interrupt vectors) and other information associated with the process virtual machine. The System/370 protection mechanism is used to prevent user-level program access to the UNIX System Supervisor. The System/370 architecture allows sharing segments among several virtual memories; as in the standard UNIX system, this facility is used to permit sharing both read-only user text and UNIX System Supervisor itself among UNIX system processes.

A program in one level communicates with the next lower level through system calls. There are two types of system calls: UNIX system calls, as defined by the UNIX System User Reference Manual, used by user-level programs to invoke the UNIX System Supervisor; and Resident Supervisor system calls, used by the UNIX System Supervisor to request certain lower-level functions of the Resident Supervisor. User-level programs never communicate directly with the Resident Supervisor. Information may be passed from a lower level to the next higher level either synchronously as return data from a system call, or asynchronously as a virtual machine interrupt (Resident Supervisor to UNIX System Supervisor) or a signal (UNIX System Supervisor to user-level program). Where available, the system takes advantage of the System/370 Virtual Machine Assist feature, which allows a user-level system call to be passed directly to the virtual machine.

2.2 Paging

As with most System/370 operating systems, the UNIX system for System/370 uses paging to manage main storage. A 16-megabyte process consists of up to 4096 pages, each of 4096 bytes; only those pages that have been allocated and referenced by the process physically exist. At any given time, these pages may be scattered through main storage and secondary (drum or disk) storage. For each process, the Resident Supervisor maintains segment and page tables, giving the main and secondary storage locations of its pages; these tables are used by the hardware when translating a virtual address to a physical main storage address. Pages are brought into main storage on demand; when an executing process attempts to reference a page not in main storage, a page fault occurs. The Resident Supervisor initiates an input operation to bring the missing page from secondary storage to main storage. The process is blocked while the page is read, and another process may be given the processor. The fact that a process may be arbitrarily blocked by a page fault while executing in the UNIX System Supervisor has ramifications to process synchronization; this is discussed in Section 2.5.

Process pages are moved out of main storage to secondary storage as necessary, on a roughly least recently referenced. The Resident Supervisor attempts to keep the "working set" of active processes-- those pages recently referenced--in main storage. All of a process' pages, including those containing the UNIX System Supervisor, are paged; a process that has been inactive for some time has no pages left in main storage. In addition, the process segment and page tables themselves can be paged and will also eventually be moved to secondary storage if the process is long inactive. The amount of permanently resident information required to represent a process is quite small, a few hundred bytes. The system also has a page migration mechanism, whereby pages of long-inactive processes may be moved from fast secondary storage (drum, fixed-head disk, or solid-state memory) to slower storage (moving-head disk).

2.3 I/O system

UNIX file systems on System/370 are in format identical to standard UNIX file systems, except that the block size has been enlarged to 4096 bytes. This block size is more appropriate to a larger system and allows us to use the paging interface described in this section. As in the standard UNIX system, I/O is blocked through a large number of block buffers, which effectively form a cache memory for recently referenced blocks. These buffers exist in shared virtual memory within the UNIX System Supervisor area. On a 16-megabyte system, we typically allocate 4 megabytes to block buffers. When a block I/O request is made to the UNIX System Supervisor, it first searches this cache for the desired block. If the block is not found, it allocates a buffer for the block and asks the Resident Supervisor to read it in.

The Resident Supervisor provides simple `read block` and `write block` primitives, which essentially provide a UNIX System Supervisor interface to the Resident Supervisor's paging mechanism. Requests for file system I/O from the UNIX System Supervisor are handled in essentially the same way as paging requests initiated by the Resident Supervisor. For example, a `read block` request simply updates the process page table. The block may not actually be read until the UNIX System Supervisor attempts to reference it, at which point a page fault occurs and the input operation is processed like a normal page-in operation. The UNIX System Supervisor may also request that I/O be initiated at the time a `read block` is executed; this is usually done to provide I/O and process execution overlap. All disks and drums in the System/370 configuration are formatted into 4096-byte records. All I/O to these devices is done through highly optimized "drivers" in the Resident Supervisor. Storage on these devices may be allocated either to the Resident Supervisor for process paging, or the the UNIX System Supervisor for file system storage.

The Resident Supervisor's `read block` primitive is used by the UNIX System Supervisor in a special way when processing an `exec` system call. Rather than reading the executable file into main storage through the buffer cache, the Resident Supervisor effectively maps the executable file into the lower part of the UNIX system process virtual address space by putting pointers to the file's disk blocks in the process page tables. As this program executes, the usual page-fault mechanism is used to read missing blocks of the executable file into main storage. The advantage of this mechanism is that only those blocks of an executable file that are actually required during execution are read into main storage.

The function and form of the character I/O system is conventional. Most drivers for character-oriented devices construct channel programs styled after System/370, and issue the Resident Supervisor `iocall` system call to execute them. All devices are known symbolically to the UNIX System Supervisor; the Resident Supervisor does the messy work of translating the symbolic address into a physical address, finding a nonbusy path to the device (including a different processor in some configurations), and initiating physical I/O. Terminal device drivers work through a special terminal interface to a front-end processor; this is discussed in Section 2.6.

2.4 Process creation

As in the standard UNIX system, processes are created by the `fork` system call; the new (child) process is created by effectively copying the calling (parent) process. In the System/370 implementation, a conventional fork would be complicated by the fact that parts of the parent process may be scattered through main and secondary storage. Since the user process may be very large (nearly 8 megabytes), a full copy could also be very slow.

Fortunately, we can again take advantage of the page-fault mechanism to avoid explicitly copying except when necessary, and to delay most of this copying so as to minimize the data actually copied at the time a `fork` is executed. When a child is created, both the child and parent's page tables are set to point to the same copy of a page--be it in main or secondary storage--with the "page fault" bit set. A private page that is "temporarily" assigned to both a parent and a child is called a multiplexed page, and a multiplexed page count, the count of processes that own this page, is kept. Subsequently, if either the parent or the child references this page, a page fault occurs; at this time the page is actually copied, and the multiplexed page count is decremented. Whenever the multiplexed count is reduced to one--either due to copying, or because the parent or child releases the page due to process death or an `exec`--the page is no longer considered to be multiplexed and may be given directly to the remaining process. In practice, this multiplexed page mechanism is quite efficient, because it implicitly takes advantage of a common UNIX system characteristic. In most cases, following a `fork` system call, the child process almost immediately performs `exec` on another program, thus discarding the data just copied by `fork`. By not copying most process data until those data are actually referenced--which, in the usual case, never happens--the System/370 `fork` executes rapidly, regardless of process size.

1.5 Process synchronization

In the standard UNIX system, process synchronization is achieved through events with associated sleep and wake-up operations. This mechanism is adequate for the usual UNIX system environment, in which processes cooperatively share a single processor. This mechanism is not sufficient for the System/370 implementation, for two reasons. First, a process on the System/370 may be arbitrarily blocked by the Resident Supervisor at any time (for example, because of a page fault), and another process be given the processor. Second, several models of System/370 are multiprocessors, with two or more identical processors sharing a common main storage and, in some cases, a common I/O configuration. In such a system, we may have two or more processes executing at the same time, possibly executing the same UNIX System Supervisor instructions. We thus need a synchronization mechanism that is indivisible on a single processor and that guarantees synchronization when simultaneously executed on a multiprocessor.

Perhaps the best known process synchronization mechanism is the Dijkstra semaphore, with associated P and V operations. A semaphore is simply a counter. When positive, it represents the number of resources available (typically, one when used for mutual exclusion); when negative, its absolute value is the number of processes waiting for the resource. The P operation is used to obtain the resource; it decrements the counter and waits if necessary. The V operation is used to release the resource; it increments the counter and awakens the (next) waiting process, if any. Semaphores have the desired indivisibility and multiprocessor-synchronizing properties, and in most cases replacing sleep and wake ups with P and V, respectively, was straightforward.

However, simply replacing existing events with semaphores is not sufficient. In the standard UNIX system, the kernel uses synchronization only where there is some possibility that it may have to give up the processor-typically to wait for an I/O operation to complete. In the System/370 implementation we must guarantee exclusive access to virtually all updates of shared system data by the UNIX System Supervisor. We thus had to identify all instances of such updates in the UNIX System Supervisor and surround them with P and V operations.

Extending process synchronization to all shared data objects in the UNIX System Supervisor was one of the more difficult parts of this implementation. This had to be done so as to guarantee the validity of the data, while avoiding the possibility of race conditions and lockouts. To minimize process blockage, we wanted this synchronization to be fine-grained-for example, to protect individual elements in an array or table, rather than simply the whole table. This led to a large number of semaphores, with rules concerning how and in what order P and V operations should be executed. Happily, the basic structure of the UNIX system kernel lent itself to this effort; very few changes in structure or program flow were made.

The System/370 instruction set does not contain P and V instructions. However, it does include a synchronizing instruction, Compare and Swap (CS), that was used in implementing P and V. The efficiency of P and V is critical; most file-system system calls execute a dozen or more of these operations. We were able to implement these operations in such a way that the Resident Supervisor is called for a P operation only if the process must wait, and for a V operation only if another process is waiting. Initially, P and V were implemented as assembler-language subroutines; subsequently they were reimplemented as inline macros. A side benefit of semaphores, especially significant on larger processors with many processes, is that only one process-the next in line-is awakened by the V operation; in essence the process executing the V passes control of the resource to the next waiting process. This differs from event synchronization, in which all processes waiting for the event are awakened by wake-up, and must again compete for the resource.

2.6 Terminals

One of the more difficult problems in making the System/370 environment look like the standard UNIX system environment occurred in terminal handling. The standard UNIX system uses a full-duplex protocol: characters typed by a user at the terminal are not displayed immediately but are sent to the processor; they are (usually) reflected back and printed or displayed. A user program may choose to process each character as it comes in ("raw mode"). Large IBM systems conventionally use a half-duplex protocol: characters are printed or displayed by the terminal as they are typed and sent to a communications controller. The characters are usually buffered here and not sent to the main processor until a special signal or character (e.g., carriage return) is typed. The UNIX system is considerably more flexible, in that special characters and associated functions can easily be defined by system or user software. However, it does imply the overhead of an I/O interrupt with each character. Some systems, such as the AT&T 3B20S computer, avoid this overhead in normal operation with a special I/O or front-end processor.

In the System/370 implementation, we wanted to provide full-duplex terminal protocol with standard UNIX system features but without character-at-a-time interrupts in the usual case. This implied the use of a front-end processor tailored to the UNIX system environment. The standard IBM System/370 communications controllers proved unsuitable for this application. However, IBM makes a minicomputer, the Series/1, with both good terminal communications facilities and a System/370 channel interface. Further, there were existing Series/1 control programs that could be used as a base for a UNIX system terminal handler. Consequently, we contracted with IBM's General Systems Division to provide a UNIX system terminal handler to our specifications. This code was delivered in late 1980, and the Series/1 is currently used for terminal handling on the System/370.

We have recently implemented a prototype front-end processor for the UNIX system for System/370 using a 3B20S system running standard UNIX System V. This implementation has a number of advantages; for example, it allows us to provide all the terminal features offered on the 3B20S computer in System V and subsequent releases. Also, it may eventually allow us to down load some frequently used character-oriented, raw-mode programs, such as screen editors, from the System/370 host. Although initially implemented on a 3B20S computer, other models in the 3B family of computers may be used. A number of such processors linked together with a System/370 mainframe could form a network of individual and group work stations, providing access to the powerful central machine as needed.

III. PERFORMANCE

One of the most interesting questions about the UNIX system on System/370 is its performance. A number of factors made the performance of the System/370 implementation unique. These factors have a considerable impact on the performance trade-offs made in the typical minicomputer implementations of the UNIX system. Coupled with the computing requirements of the large system-development task for which it was first used, the 5ESS local digital switch, these factors determined the capacity of the System/370 implementation. The scale of the system also demands longer-range capacity forecasting than typically applied in minicomputers. The following sections discuss these points in more detail.

3.1 Unique factors

The UNIX system on the larger models of System/370 line, such as the IBM 3081K, increases by over an order of magnitude the scale and scope that the operating system must manage. Numbers of processes, I/O buffers, file descriptors, i-nodes, and other system resources are measured in hundreds or thousands rather than tens or hundreds as on minicomputers.

One of the earliest concerns about a UNIX system implementation for large processors was its ability to "scale"; that is, were there inherent characteristics of the UNIX system and its algorithms that limited its implementation on large machines? Happily, we found that in most cases the straightforward algorithms that implement the resource policies of the UNIX system perform quite well on this scale, leading one to question the complex algorithms more typically employed in large operating systems. In a few cases the standard algorithm was replaced for efficiency; for example, the standard UNIX system linear search of the block buffers was replaced by a faster search based on hashing. The major area where scale appears to have altered the character of the UNIX system is that of resource limitations on individual users or processes. The impact of looping processes and file space consumers is more widespread, and the cause is more elusive than in smaller systems. Efforts to detect and correct these types of problems have substantial benefits in the System/370 environment.

Additional resources available on a mainframe, such as multiple central processors, powerful autonomous I/O channels, fast peripherals such as drums and solid-state mass stores, large amounts of main storage, and communications front-end processors greatly enhance the throughput of the UNIX system. In particular, the dramatic increase in I/O bandwidth coupled with the use of ample main storage for the disk block cache avoids the I/O-bound behavior typical of smaller UNIX systems. The increased main storage and efficient paging capability increase the number of dispatchable processes and reduce idle time. The front-end communication processors buffer the central processor(s) from character at a time I/O unless required by the application (the so-called raw mode).

A number of adaptations of the UNIX system that take advantage of the characteristics of the mainframe also enhance performance. The larger block size used (4096 bytes versus 512 or 1024 bytes in smaller machines) reduces the overhead in I/O activities. To avoid the dramatic loss of usable space that small files and directories would cause with 4096-byte blocks, the concept of large-block/small-block files was introduced. Files of less than 493 bytes are stored directly in the corresponding i-node. As a side effect, once the i-node for a small block file is read, no further disk access is required to retrieve the file contents. This proves to be particularly beneficial for shell scripts, which are commonly used and often quite small, as well as for small directories. In keeping with the scale of the mainframe and the development being done on them, the file size limit on System/370 is currently 16 megabytes. This reduces the need to create and process multiple files in applications such as databases, which require very large files.

3.2 Performance trade-offs

As a result of the factors cited above, the typical performance tradeoffs on a System/370 machine are different from those for the minicomputer UNIX systems on which most of the current UNIX system programs were developed. Many UNIX system programs make extensive use of temporary files for even modest amounts of data. Some tools, such as the C compiler, were divided into multiple processes interconnected by temporary files to work around memory limitations imposed by early UNIX system hosts such as the PDP-11 computer. The increased I/O bandwidth and the fact that many small temporary files remain fully in the disk block cache reduces the impact of the widespread use of temporary files, but in areas where such files have been eliminated, the performance gains have been impressive. In general, a shift in emphasis from temporary files toward greater use of main memory takes advantage of the additional spectrum available and allows the efficient paging mechanism to dynamically manage data that the programmer had previously explicitly and statically managed. Despite the trend toward increased use of memory, the average process still requires less than 200 kilobytes of the 8-megabyte user space.

3.3 System capacity

To determine system capacity of the UNIX system on System/370 machines relative to minicomputers such as the PDP-11/70, VAX-11/780, and 3B20S computers, a set of scripts of typical software development command mixes were developed and applied to differing UNIX system configurations. Results indicated that the IBM 3033AP configuration first put into production was equivalent to several VAX-11/780 or PDP-11/70 systems. Tuning of the VAX, 3B20S, and System/370 computers has varied these ratios over time, but the overall order of magnitude spread has been maintained. Use of the newer IBM 3081K processor has increased capacity by 50 percent, and evolution to the IBM 3084Q promises larger gains. In actual operation a single large system obtains further efficiencies over the equivalent number of smaller systems in terms of networking, operation, and administration. In general, we have found that highly processor-intensive work loads, or work loads requiring a lot of parallel file system I/O, run relatively better on the large System/370 machine; work loads characterized by many short interactions, context switches, and character-oriented I/O run relatively more poorly.

Typical operational parameters of an IBM 3033AP are 150 simultaneous users (upwards of 200 have been observed), 600 active processes (upwards of 1000 have been observed), 90-percent CPU usage on both processors, and 10- to 20-percent usage of the I/O channels.

IV. INITIAL APPLICATION

4.1 Porting the application software

In early 1981 a production UNIX system was running on an IBM 3033AP in the Bell Laboratories Indian Hill Computation Center. The next step was to port the application software tools of the 5ESS switch development environment from the PDP-11/70 computers to the 3033AP. Over 300 tools, written in both C and shell command language, were identified and examined. After careful study, almost half of the tools were found to be little-used and were eliminated as candidates for porting to the 3033AP. The C programs required recompiling to generate objects that would run on a 3033AP; in general, they complied without problems. The shell scripts were carried over with almost no problems. Regression tests were used on the various C compilers to test all the compiler, assembler, and loader functions, and other programs were unit tested. System testing, which consisted primarily of generating the system software for the 5ESS switch, was then done.

In general the porting went very smoothly, with only minor problems. To the application program developer and user, the System/370 appeared to be the same as the UNIX system on the minicomputers that they were using. The effort to port the application tools was small and again proved the strength and computer independence of the UNIX operating system and the associated application programs.

4.2 User migration

After testing the UNIX operating system and the application software tools, the users were migrated from the PDP-11/70 computers to the 3033AP. To avoid a significant impact on the development of the 5ESS switch, a gradual rather than a flash migration was selected. The 3033AP was networked into the nine PDP-11/70s and appeared as the tenth system. This allowed moving a subset of the users to the 3033AP but required continuing the multicomputer procedures to generate the software for the 5ESS switch. About 10 percent of the users were moved on a weekend every two weeks. This allowed the staff that was in charge of the migrations to work with these users, identify any special needs, and solve the small number of problems that came up with each group. The users experienced no problems with the use of the new machine because they saw the same user interface as before. This allowed the migration to proceed without the cost of any user education or any lost time as the users learned the new system.

4.3 Reliability

The combination of complex hardware with an attached processor configuration and the Series/1 front-end processor plus the three software packages (IBM Resident Supervisor, UNIX System Supervisor, and Series 1 Terminal Handler) all interacting initially produced an availability of 80 percent. Even with 80 percent availability the project made progress faster than ever with the addition of a large concentrated processor. By the final migration the availability was improved to the 95-percent range. In the next six months the availability was improved to the 97- to 98-percent range, where it has stabilized. This is the same range as the mature TSS/370 operating system running on similar hardware. While there were some early problems, they were much less than we had ever experienced in transferring a project to a new operating system and the reliability that is associated with very mature operating systems was reached more quickly than we had ever experienced.

4.4 Multiple System/370 environment

As the development project for the 5ESS switch continued to grow, additional System/370 machines were added to the environment. The multiple PDP-11/70 software was ported to the IBM environment, and successful multimachine operation was again in place. The current environment includes IBM 3033AP, 3033UP, and 3081K systems. The first application of a IBM 3081K processor with approximately 50 percent more throughput than the 3033AP was in early 1983. This new system was brought up with the UNIX operating system and the applications tools with no changes. From the first day it displayed the reliability of a mature system.

4.5 Experience summary

The UNIX operating system with the Programmer's Workbench software has proven to be an excellent system to support software development. Our experience in developing the software for the 5ESS switch has shown that there is a limit to the size of a software project that can be supported on minicomputers. Up to now the UNIX operating system was not available on the large mainframe computers that are necessary to provide the computing resources needed by a large project. With moving the UNIX operating system to System/376 class mainframe systems, large projects can now take advantage of the UNIX operating system and its tools.

V. CONCLUSIONS

The UNIX system for System/370 has now been in production service for over two years, primarily in support of the development project for the 5ESS switch. The growth in the number of systems and the diversity of the IBM processors used (3031AP, 3033U, 3033AP, 3081K, and 4341) both testify to the success of the concept of a UNIX system implementation for mainframe computers. Several innovative features of the System/370 implementation, such as the use of semaphores for process synchronization, have been found useful in other UNIX system implementations.

The proposal to implement the UNIX system on a large mainframe computer was initially met with some skepticism. This may have been in part a result of the "small is beautiful" argument, and the feeling that operating systems for large mainframes were themselves necessarily large, complex, and difficult to use. We hope that the System/370 implementation has helped to demonstrate that this is not true. The availability of the UNIX system on a large mainframe has again raised the issue of small versus large machines; e.g., should an installation buy several small systems, or would one large mainframe be better? There is, in fact, nothing inherently better about either large or small systems; the decision should be based on the user's requirements, the character of the work load, and the overall cost.

The UNIX system is the only operating system available that runs on everything from one-chip microcomputers to the largest general-purpose mainframes. While this represents at least a two-orders-of-magnitude range in power and capacity, functionally the environments are the same; most programs that execute in one environment will execute in the other without change. The ability of the UNIX system to gracefully span the range from microcomputers to high-end mainframes is a tribute to its initial design over a decade ago and to its careful evolution.

REFERENCES

1. S. C. Johnson and D. M. Ritchie, "UNIX Time-Sharing S stem: Portability of C Programs and the UNIX System," B.S.T.J., 57, No. 6 stem: 1978), pp. 2021-48.
2. IBM System/370 Principles of Operation, Ninth Edition (October 1981), GA22-7000-8, IBM Corporation.
3. W. B. Smith and F. T. Andrews, Jr., "No. 5 ESS - Overview," Bell Telephone Laboratories; International Switching Symposium, Montreal, Canada, September 1981.
4. T. A. Dolotta, R. C. Haight, and J. R. Mashey, "UNIX Time-Sharing System: The Programmer's Workbench," B.S.T.J., 57, No. 6 (July-August 1978), pp. 2177-2200.
5. IBM System/360 Time Sharing System: System Logic Summary, Third Edition (June 1970), GY28-2009-2, IBM Corporation.