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

UNIX Time-Sharing System:

The Network Operations Center System

By H. COHEN and J. C. KAUFELD, Jr.
(Manuscript received January 27, 1978)

The Network Operations Center System (NOCS) is a real-time, multiprogrammed system whose function is to survey and monitor the entire toll network. While performing this function, the NOCS supports multiple time-shared user work stations used to interrogate the data base maintained by the real-time data acquisition portion of the system. The UNIX operating system was chosen to support the NOCS because it could directly support the time-shared user work stations and could, with the addition of some interprocess communication enhancements and a fast access file system, support the real-time data acquisition. Features of the UNIX operating system critical to data acquisition include signals, semaphores, interprocess messages, and raw I/O. Two features were added, the Logical File System (LFS), which implements fast application process access to disk files, and the Multiply Accessible User Space (MAUS), which allows application processes to share the same memory areas. This paper describes these features, with emphasis on the manner in which UNIX operating system features are used to support simultaneously both real-time and time-shared processing and on the ease with which features were added to the UNIX operating system.*

I. INTRODUCTION

This paper explains how the Network Operations Center System

* UNIX is a trademark of Bell Laboratories.

(NOCS) makes use of the UNIX* operating system to perform its functions. Thus, while some explanations of the NOCS and its processes are given, the primary intent of the NOCS explanations is to motivate the discussion of UNIX operating system features and modifications.

The paper is organized into three sections. The first deals with the NOCS, giving a simplified overview of the system and an analysis of the critical functions and the features needed in an operating system to implement those functions. The second section details a few of the features added to the standard UNIX operating system and the functions these features are intended to fulfill. The third section discusses the advantages which the UNIX environment offered to the NOCS project during development.

II. NOCS DESCRIPTION

2.1 Overview

2.1.1 Network management

Network management of the telephone system concerns itself with maximizing telephone network utilization. It is performed in a hierarchy of levels with responsibilities divided among local, regional,† and North American network management centers. At all levels, network managers continuously monitor the volume of telephone traffic being placed on trunk groups and offices within their respective spheres of responsibility. Whenever the volume begins to exceed the design capacity of their own portion of the network, network managers attempt to find out-of-chain alternate routes through other portions of the network for that telephone traffic. If no alternate route can be found, or if insufficient capacity is available in the network to handle the traffic presented, network managers have the option to cancel a portion of the load via code blocks and other traffic-reducing controls, since network call-completing efficiency decreases when overloaded. To perform these functions, network managers require large amounts of up-to-the-minute traffic data that have been analyzed, summarized, and displayed in a concise, filtered manner.

* UNIX is a trademark of Bell Laboratories.
† The telephone switching network is partitioned into 12 segments known as switching regions.

2.1.2 NOCS role

Network managers at the Network Operations Center operated by AT&T Long Lines in Bedminster, New Jersey have overall responsibility for the entire North American toll telephone network. These responsibilities include:

(*i*) Monitoring the status of all major toll-switching machines and their interconnecting trunk groups.
(*ii*) Coordinating the efforts of other network management centers in charge of smaller geographic portions of the network.
(*iii*) Detecting interregional trunking problems and recommending reroute control solutions to the other network management centers.
(*iv*) Distributing network performance and status information to appropriate telephone company management personnel.

The NOCS provides the network managers with many tools to aid them in fulfilling these responsibilities and relieves them of much of the data uncertainty and manual effort previously associated with their jobs. Application processes in the NOCS collect data, maintain a visual display of network status, analyze data looking for network problems, suggest possible control actions, and provide the ability to make a wide variety of complex data inquiries about the network. These processes drive a wall display* that gives a visual overview of the state of the network, printers that report details of network activity, and 10 or more cathode ray tube (CRT) terminals that are used to examine problems in detail, to update the data base, and to administer the system.

2.1.3 Data network

The data transfer point (DTP)† is at the hub of a star-configured computer network that has EADAS/Network Management (E/NM)‡

* The wall display contains approximately 8000 binary state indicators and is about 3 meters tall by 20 meters long.
† The DTP, which utilizes a Digital Equipment Corporation PDP 11/70 minicomputer, is a UNIX-operating-system-based system designed to broadcast network management data among all E/NM systems and the NOCS according to routing information contained in the data messages.
‡ The Engineering and Administrative Data Acquisition System/Network Management system (E/NM) is another Digital Equipment Corporation PDP-11/70 minicomputer system running under the UNIX operating system. It is designed to monitor and control both local and toll traffic in a geographical subset of the telephone network —

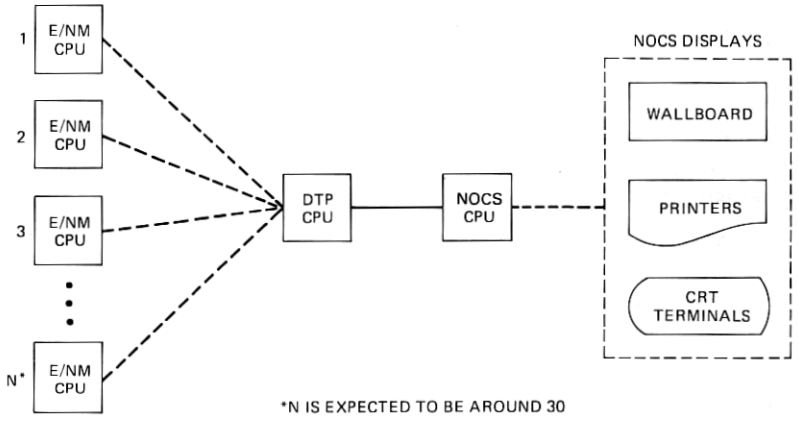

Fig. 1—Network management data network.

systems and the NOCS as the nodes (see Fig. 1). The NOCS, which executes on a Digital Equipment Corporation PDP-11/70 minicomputer system, is physically co-located with the DTP. This network is synchronized in time to utilize telephone traffic data accumulated periodically (every 5 minutes) by data collection systems. When an E/NM detects a problem or control on a trunk group or office of potential concern to the NOCS, the data describing that problem or control are sent to the DTP which passes them on to the NOCS. In addition, this network is used to pass reference data describing the telephone network configuration to the NOCS.

2.1.4 NOCS data

During the first 150 seconds or so of each 5-minute interval, the NOCS can receive as many as 6000 dynamic data messages, i.e., data describing the current state of the network, through the link connecting the NOCS and DTP computers. About 2000 of these messages concern the trunk status data needed continuously by the NOCS in order to find spare capacity for alternate routes. Most of the rest are trunk group or office data messages which occur only when a trunk group or office is overloaded beyond its design capacity. Each message contains identity information and several processed traffic data fields, which are retained for several intervals in files in the NOCS data base. Dynamic data messages which give trunk group control, code block control, and office alarm status are

such as a large metropolitan area, a state, telephone operating company, or switching region. It is expected that 30 or more E/NMs will blanket the U.S. telephone network by the early 1980s.

received asynchronously in time by the NOCS, and are retained in the NOCS data base until a change in state occurs.

The telephone network configuration of interest to the NOCS is specified by reference data describing approximately 300 toll switching offices and up to 25,000 of their interconnecting trunk groups. All the trunk group reference data needed by the NOCS are derived algorithmically from the reference data base of the E/NMs and are transmitted to the NOCS via the DTP. Hence, reference data messages that contain updates to the network configuration are received at irregular intervals from E/NM systems.

2.1.5 NOCS data base

The NOCS data are arranged into a data base consisting of several hundred data files containing all the reference data necessary to describe the network configuration and the dynamic data needed to reflect the current state of the network. Some of the files are record-oriented, describing all the information about individual data base entities such as an office or trunk group. Others are relational in nature, containing sets of related information such as all the trunk groups between geographic areas or all the trunk groups with the same type of problem condition. This arrangement allows complex data inquiries to be answered quickly by combining relations with a standard set of operations. It also allows per-entity data to be accessed very quickly by a simple indexing operation into a file.

2.2 NOCS design

2.2.1 Philosophy

The UNIX operating system was selected as the basis for design to take advantage of prior experience with that system. During the design phase, every attempt was made to use existing UNIX operating system features to implement the required NOCS functions and to minimize the number of modifications necessary to the UNIX operating system. As a result of this philosophy, the final set of features needed in the operating system by the NOCS included only two features not in the standard UNIX system. If the UNIX operating system modifications had turned out to be too extensive to be implemented locally, another operating system would have been investigated.

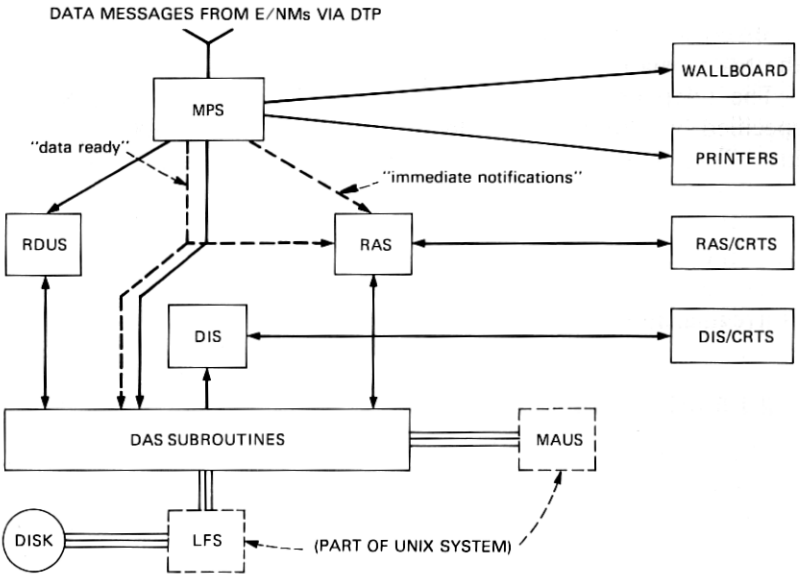

Fig. 2—Major NOCS subsystems.

2.2.2 NOCS subsystems

The application software for the NOCS is organized along the lines of functional requirements into the major subsystems listed below (for graphic representation, see Fig. 2). Each of these subsystems consists of one or more UNIX processes.

(*i*) The Message Processing Subsystem (MPS) receives all incoming data from the DTP. Some incoming data from the DTP are reference data messages which are not handled in real-time. These reference data messages are passed to the Reference Data Update System (RDUS), a background process that maintains the NOCS reference data base. The remainder, dynamic data messages, are handled by the MPS and are entered into the NOCS data base. If an "immediate notification" request has been made for a data item by some other NOCS subsystem, an interprocess message containing that data item is sent to that subsystem. When all the dynamic data for an interval have been received, the MPS notifies all interested subsystems of the availability of new data, updates the data indirection pointers to make the new data available, updates the wall display to indicate the latest network problems, and begins printing the latest data reports.

(*ii*) The Reroute Assist Subsystem (RAS) analyzes new data each 5-minute interval to determine if there are any interregional final trunking problems in the network that are possible candidates for traffic reroutes. If any are found, RAS looks through the available trunk data to determine if spare capacity exists to relieve the problem. Any problems and suggested solutions are displayed to the network managers through CRT terminals. The RAS also requests the MPS to notify it immediately if any further data are received relating to an implemented reroute. If such notification is received, the RAS immediately examines the data given it by the MPS to determine if any changes should be made to that reroute.

(*iii*) The Data Inquiry Subsystem (DIS) provides the network managers with access to the NOCS data base from CRT work stations. From any of these work stations, a manager can display network data in a variety of ways. Each way presents the data from a unique perspective in a rigorously formatted and easily interpreted manner. In addition, CRT data entry displays are also used in maintaining the nontrunking portions of the reference data base.

(*iv*) The Reference Data Update Subsystem (RDUS) processes all changes relating to the configuration of the network. Inputs to this system can come from the E/NMs via the MPS via the DTP, CRT displays or as bulk input through the UNIX file system. It uses these inputs to create and maintain the reference data base needed by the MPS, RAS, and DIS to effectively interpret, analyze, and display the dynamic data.

(*v*) The Data Access Subsystem (DAS) handles all requests for information from the NOCS data base. The DAS consists of a large set of subroutines that provide access to the NOCS dynamic and reference data files. Hence, in the UNIX sense, DAS is not a process but a set of routines loaded with each UNIX process that accesses the NOCS data base. Because multiple processes simultaneously need quick access to the same files for both reading and writing, the DAS maintains a large area of common data and synchronization flags, which is shared by all NOCS processes using the DAS.

2.2.3 Operating System Problem Analysis

2.2.3.1 File system requirements. The initial processing of data

messages by the MPS involves a trunk group identity translation, a possible wallboard indicator translation and storage of the data items contained in the message. A simple analysis of the relationship among message content, the expected message volume, and the data display requirements reveals that the NOCS must be able to do at least 12,000 translations on the set of incoming data messages and place them into the correct disk files in an elapsed time of about 150 seconds. Potentially, then, a very large number of disk accesses are possible unless careful attention is given to the algorithms and data layouts used. Thus, a data storage mechanism is required with as little overhead as possible from the point of view of system buffering, physical disk address calculation, and disk head positioning. This simple analysis does not take into account the substantial number of disk accesses necessitated by DIS, RAS, and other NOCS background processes.

In addition, analysis of data inquiry displays reveals that some DIS processes would need simultaneous access to more data files than the UNIX file system allows a process to have at one time. Therefore, in order to hold response times down, some mechanism for overcoming this limitation without numerous time-consuming opening and closing of files is necessary.

2.2.3.2 Scheduling requirements. The application processes in the NOCS impose three types of scheduling criteria on the UNIX operating system. First, the MPS and RAS are required to analyze the incoming data in "near" real-time to provide network managers with timely notification of potential problems and recommended solutions. Since the MPS is responsible for collecting and processing incoming data, it must execute with enough frequency and for long enough periods to prevent data overruns and/or delays. Second, the CRT work stations, at which network managers interact with the DIS, have classical time-share scheduling needs. Third, background programs, exemplified by the processing of reference data messages by RDUS, require neither real-time nor time-share scheduling. Processes of this third type can be scheduled to run whenever no processes of the real-time or time-share variety are waiting to run.

2.2.3.3 Interprocess communication requirements. Several types of interprocess communications mechanisms are needed by NOCS subsystems. First, the MPS must send interprocess messages to other NOCS processes upon reception of "immediate notification" requested data. These data must be passed quickly but are not large in volume. The use of an interprocess message facility implies that the process identifications (process IDs) assigned by the UNIX

operating system* be communicated among processes. Another mechanism mentioned in the MPS description was the need to be able to notify other processes of the availability of new data. This mechanism must be able to interrupt the actions of the target process so that any necessary processing can occur before resuming the interrupted activity. Last, it was also foreseen that the implementation of the DAS would require multiple processes to share their knowledge of the state of files in the data base and would require a mechanism for synchronization and protection of these files.

2.2.4 Operating system problem solutions

2.2.4.1 File system. The file system requirements led to the conclusion that the standard UNIX file system was inadequate for the NOCS. However, the raw I/O facility in the UNIX operating system has the following features:

(*i*) Control of data block placement within the raw I/O area.
(*ii*) Transfer of data directly from disk to user buffer.
(*iii*) Access to a large contiguous disk area through a single UNIX file.

These features provide precisely the capabilities required by the NOCS in a file system. Hence, they were used as the foundation for a new type of file system, known as the Logical File System (LFS), which was added to the UNIX system.

The LFS controls logical file to physical disk address mapping and the allocation of space on the disk. The LFS can be requested by the user to read or write 512-byte sectors of a file, create or delete a file, or copy one file to another. It keeps all files contiguous to simplify logical to physical address mapping and minimize disk head movement. Also, it transfers data directly from disk to user buffer areas.

The entire NOCS data base is implemented using the LFS. Since all access to data from NOCS processes is through DAS subroutines, the DAS has total semantic responsibilities for the contents of the files which are in the LFS. The DAS remembers what NOCS data are in which file, the size of the file in bytes, and the current usage status of the file.

2.2.4.2 Scheduling. The NOCS scheduling requirements are such that the standard UNIX facilities can handle them. All real-time

* The process identification is the only way of uniquely identifying a job once it has been started by the UNIX system. The message mechanism uses process identifications as its addressing mechanism.

processes are given a base priority that is higher than any time-share process. Thus, in the case of competition for the processor, the real-time processes will be scheduled first. Within the real-time priority range, the MPS is given the highest priority to ensure that it will always be able to process the incoming data. Within the time-share priority range, the CRT work stations assigned to reroute assist interaction are given the highest priority. Finally, background processes, like RDUS, are given lower priority than any time-share process to ensure that background processes will only be run if no other work needs to be performed.

2.2.4.3 Interprocess communications. The final design for the NOCS system relies on the following interprocess communication mechanisms:

(*i*) A mechanism for communicating process identifications.
(*ii*) An interprocess message facility for passing small amounts of data between unrelated processes.
(*iii*) An interrupt mechanism for interprocess notifications of events.
(*iv*) A synchronization/resource protection mechanism.
(*v*) A mechanism for sharing large amounts of data between processes.

Given the existence of item (*i*), the UNIX interprocess message facility can handle item (*ii*) and the UNIX signal facility can handle item (*iii*). However, items (*i*), (*iv*), and (*v*) required additions to the UNIX system. Item (*iv*), the synchronization/protection mechanism, was solved by expanding the semaphore capability of the UNIX system. Semaphores existed in the UNIX system, but processes were restricted to five semaphores, and about 400 were needed by the DAS simultaneously to effectively use the LFS capability. One way in which item (*v*), sharing of data, could be handled by the standard UNIX system would be to establish a file (either in LFS or the UNIX file system) whose contents were read by each process when necessary. In addition, a series of semaphores would have to be established so that processes could have exclusive access to this file for the purpose of changing it. A system for data sharing of that design would have many problems in terms of simplicity, synchronization, and speed, so it was decided to make a major addition to the UNIX operating system known as MAUS, or Multiply Accessible User Space. MAUS allows processes to directly share large portions of their data space. MAUS also provides a solution for item (*i*).

2.2.4.4 Other. A variety of other problems were encountered, most of which were solved without any modifications of the UNIX operating system. However, the following other additions were made to the operating system:

(*i*) A DA11 UNIBUS link driver for high-speed interprocessor communications. This is used for the DTP-to-NOCS communications link.
(*ii*) A DH11 asynchronous line multiplexer driver for half-duplex DATASPEED® 40 terminals.
(*iii*) Disk I/O priorities so that the priority of a disk request matches the priority of the process needing the disk.

All the above modifications were necessary, but considered sufficiently straightforward to need no further explanation.

III. UNIX FEATURE ADDITIONS

The Logical File System and the Multiply Accessible User Space features were implemented for the NOCS. These features are now available and being used by other UNIX-system-based application systems needing real-time operating system features.

3.1 LFS (Logical File System)

The LFS is a mechanism that enables processes to use the raw I/O facility in the UNIX operating system without having to manage the disk space within the disk area reserved for raw I/O. It establishes a file system oriented around 512-byte blocks within which it can create, write, read, and delete files.

3.1.1 Overview

The file system which the LFS provides sacrifices a number of features of the standard UNIX file system for simplicity of implementation. For instance, the standard UNIX file system provides access protection at the individual file level; in the LFS, access protection is only provided once for the set of files managed by the LFS. Another difference is that file names in the standard UNIX file system are character strings which can be descriptive of file contents; in the LFS, file names are numbers.

The important features of the LFS are listed below.

(*i*) Treatment of the LFS as a single UNIX file which, when opened, allows access to all LFS files.

(*ii*) File names that are indices into an array which lists the starting block and size of each file, thereby minimizing the time required to "look up" the physical mapping of a file.

(*iii*) Contiguous space allocation for all files, thereby minimizing the time required to copy file data into memory.

(*iv*) Integrated file positioning with read or write, thereby eliminating separate file positioning system calls which are necessary for accessing normal UNIX files.

(*v*) Adherence to the UNIX principle of isolating the application processes from the vagaries of the physical device — with the restriction that any physical device used for the LFS must appear to have 512 bytes per block.

In order to access the files managed by the LFS, the unique UNIX file name associated with the LFS must be opened using the special routine **lfopen**. The routines **lfcreate**, **lfwrite**, **lfread**, and **lfdelete** then can be used by processes to create, write, read, and delete files within the LFS. Each of these routines expects the LFS file number as an argument. In addition, **lfcreate** expects to be passed the size of the file being created; **lfwrite** and **lfread** expect a data buffer address, data buffer size, and starting position in the file.

The LFS has one additional feature which the NOCS software uses. The **lfswitch** routine takes two LFS file numbers as arguments and switches the physical storage pointers for the files. This feature enables files to be built in an offline storage area and then be quickly switched online; it is especially useful during data base updates.

3.1.2 Implementation

A restriction existed in the UNIX operating system which had to be eliminated before the LFS could be effective. The interface between the LFS and the disk is through the raw I/O routine, **physio**. **physio**, in the standard UNIX operating system, allows only one raw I/O request to be queued for each device; since almost all NOCS processes make raw I/O requests via the LFS, this restriction would have resulted in a severe bottleneck. The remedy was to allow **physio** to queue raw I/O requests for different processes by providing a pool of raw I/O headers analogous to the pool of system buffers available for standard UNIX file I/O.

One code module containing the LFS routines was added to the

UNIX operating system. Of course, the module containing **physio** was modified as outlined above. In addition, the **physio** modification requires minor changes in several device-handling routines. Also, several data structures were modified to allow for the pool of raw I/O headers and to define the LFS file system structure. In total, the modifications necessary to install the LFS required about 300 lines of C code to be added or modified.

3.2 MAUS (Multiply Accessible User Space)

MAUS (unpublished work by D. S. DeJager and R. J. Perdue) is a mechanism that enables processes to share memory. It is not necessary for the general time-sharing environment, but for multiprogramming real-time systems such as NOCS it is almost essential.

3.2.1 Overview

MAUS consists of a set of physical memory segments which will be referred to as MAUS segments; a unique UNIX file name is associated with each MAUS segment. A MAUS segment is described to the system by specifying its starting address (a 32-word block number relative to the start of MAUS) and its size in blocks (up to 128 blocks or 4096 words). The physical memory allocated for MAUS starts immediately after the memory used by the UNIX operating system and is dedicated to MAUS. Any process may access MAUS segments.

A process may access a MAUS segment in two ways. The preferred MAUS access method is to make the MAUS segment a part of the process's address space by using a spare segmentation register. This method can be used by any process which has at least one memory segmentation register left after being loaded by the UNIX operating system.* If the process has no free memory segmentation registers, then access to the MAUS segment may be obtained under an alternate method which uses the standard file access routines such as **open**, **seek**, **read**, and **write**. The alternate method is slower than the preferred method and has potential race problems if more than one process tries to write data into a MAUS segment.

* Each process has a fixed number of memory segmentation registers available for its use. For processes running on a Digital Equipment Corporation PDP-11/70 under the UNIX operating system, eight memory segmentation registers are available for mapping data and MAUS. Each segmentation register is capable of mapping 4096 words, i.e., one MAUS segment. One of these registers is always used for the process stack and at least one other is used for the data declarations within the process. Thus, a maximum of six segmentation registers are available for accessing MAUS.

3.2.1.1 Preferred access method. To access a MAUS segment by the preferred method, a process must first obtain a MAUS descriptor using the MAUS routine **getmaus** in a manner similar to the standard UNIX **open**. The UNIX file name associated with the MAUS segment and the access permissions desired are given in the **getmaus** call. **getmaus** makes the necessary comparisons of the access desired with the allowable access for the process making the call and returns either an error or a MAUS descriptor which has been associated with the requested MAUS segment. A process is allowed to have up to eight MAUS descriptors at the same time. When a valid MAUS descriptor is given to the **enabmaus** routine, a virtual address, which may be used by the process to access data within MAUS segment associated with the MAUS descriptor, is returned. This virtual address is also used to detach the MAUS segment with the **dismaus** routine. The MAUS descriptor can be deallocated using the **freemaus** routine. Obtaining a MAUS descriptor is very slow relative to attaching and detaching MAUS segments; thus processes which cannot simultaneously attach all the MAUS segments they need to access can still rapidly attach, detach, and reattach MAUS segments using MAUS descriptors. Any number of processes can have the same MAUS segment simultaneously attached to their virtual address space.

3.2.1.2 Alternate access method. To use the alternate access method, the UNIX file name associated with the MAUS segment desired is **open**ed like any normal UNIX file. The file descriptor returned can be used by the **read** or **write** routines to access the MAUS segment as if it were a file.

3.2.2 Implementation

One code module containing the MAUS routines and some minor modifications to several existing functions within the UNIX operating system were all that was necessary to install MAUS. In addition, several minor modifications were made to UNIX data structures to store MAUS segment descriptions and MAUS descriptors. In total, less than 150 lines of code were added or modified. In the final analysis, the most difficult part of the MAUS implementation was arriving at a design which was compatible with existing interfaces within the UNIX operating system.

IV. STANDARD UNIX FEATURE USAGE

The standard UNIX system provides a complete environment for

the entry, compilation, loading, testing, maintenance, documentation, etc., of software products. It is impossible to categorize all the ways in which this environment aided in the design and development of the NOCS; however, a few examples are illustrated here.

4.1 Development and testing

The same version of the UNIX operating system is used both for NOCS development in the laboratory and for the NOCS application. In fact, because of the versatility of the UNIX system and the design of the NOCS software, most of the NOCS system is left running continuously during software development. New versions of NOCS subsystems can be installed without the need for restarting the entire system. In essence, a continuous test environment exists so that developers can integrate their programs as soon as module testing is completed without having to schedule special "system test" time.

4.2 Software administration

The NOCS software is maintained in source form on two UNIX file systems. One of these file systems is mounted read-only and may not be modified by software developers; the other initially contains a copy of the first and is used for software development. Upon completion of a successful development milestone, an updated version of the software is moved to the read-only file system by a program administrator and a new development file system is created. Standard UNIX utilities are used to keep track of changes between the read-only and development file systems.

In order to generate the NOCS binary from the source, UNIX shell procedures have been developed. There is one "build" procedure for each NOCS subsystem. The procedures are part of the NOCS software and are administered in the same manner as the rest of the NOCS software. The structural similarity between the read-only and development file systems allows the complete testing of software "build" procedures before they are copied to the read-only file system.

4.3 System initialization

The standard UNIX init program is used to start the NOCS functions. Each NOCS process is assigned to a run level or to a set of run levels. When init is told, by an operator, to enter a particular run

level, the NOCS processes assigned to that run level are started. The assignment of processes to run levels is made in such a way that critical NOCS processes may be isolated both during development and in the field for testing.

4.4 Documentation

All NOCS documentation is done under the UNIX system. This documentation consists of a user's manual which describes the inputs and outputs for the system and a developer's guide which is a description of the NOCS software. The **nroff** UNIX program along with NOCS-developed **nroff** macro packages is used to format the documentation for printing. The text is entered using the UNIX **ed** program and is stored in standard UNIX files.

V. CONCLUSIONS

The NOCS system has real-time multiprogramming requirements that make operating system demands very much counter to the basic UNIX time-share philosophy. However, these demands were met quite readily with some feature additions because of the adaptability and generality inherent in the UNIX operating system. The available scheduling parameters were flexible enough to handle the three kinds of scheduling demands; "near" real-time, time-share, and background, imposed on the UNIX operating system by the NOCS. The interprocess communication mechanisms were rich and varied enough that only one addition was necessary to provide all the features needed by NOCS. The UNIX operating system was modular enough so that a completely new kind of file system was interfaced in a very short time with almost none of the timing bugs that might be expected in a typical operating system. The total UNIX environment provided software tools to support the complex development effort needed to implement the NOCS. Finally, the use of the same operating system for both development and application certainly minimized friction between the coding and testing phases of development and allowed the smooth integration of all system functions.