Operating system design considerations for the packet-switching environment*

by DAVID L. RETZ
Speech Communications Research Laboratory, Inc.
Santa Barbara, California

INTRODUCTION

One of the striking developments in computing and communication technology during the past decade is reflected in the evolution of packet-switching computer networks.[1,2,3] Packet-switching communication techniques allow dynamic allocation of a set of communication resources (circuits) so that they may be flexibly shared among a number of autonomous processors. Implementation of such packet-switching networks has required many design decisions, such as the choice of network topology, routing strategies, and the establishment of conventions, or protocols, for information interchange between network resources.

This paper is concerned with the design requirements of Host operating systems: those systems whose primary business is the management of computing resources rather than communication resources. Low-level communication tasks such as routing fall outside the realm of the Host responsibilities discussed here and are performed by means of a sub-network of small computers dedicated to the task of packet-switching. In the ARPANET these computers are called Interface Message Processors, or IMPS, and use packet-switching techniques to communicate via 50-kilobit common carrier circuits. Each IMP provides up to four high-speed synchronous serial ports to which Hosts connect using special-purpose Host-IMP interfaces.[4]

Packet-switching network environments place special requirements on the design of the connected Host operating systems. Attachment to the ARPANET, for example, has required a number of additions or modifications to existing operating systems. There are certain structural features which must be incorporated in system design in order to facilitate effective use of distributed computing resources. We begin by examining a few of these features.

IMPLICATIONS ON HOST SYSTEM ARCHITECTURE

Sharing of distributed resources is made possible by the cooperation of distributed processes. The notion of process has been widely used in operating system structures[5,6] in order to provide a modular representation of autonomous, event-driven computational tasks. Specification and implementation of protocols—well-defined conventions by which processes communicate—has allowed resource sharing to occur in a non-homogeneous environment. A process might utilize a remote resource such as a disk file, for example, by transmitting a prescribed command to a remote process which interprets and carries out the hypothetical command: "read the tenth record of disk file XYZ and transmit back its contents."

A layered structure of protocols has evolved to make possible network-wide sharing of ARPANET resources. The Host-Host protocol rests at the foundation of this structure, providing a mechanism by which processes in the network may communicate.[7] A number of higher level protocols make use of Host-Host protocol to perform function-oriented tasks.[8] For example, the Telnet protocol provides terminal access to remote interactive systems on the network, and the File Transfer Protocol allows files to be copied from one site to another.

The standard ARPANET Host-Host protocol creates inter-process communication (IPC) channels, or "connections," at the request of Host processes, through an exchange of special control messages between the Host operating systems. In general, this facility has been provided by the implementation of a set of procedures, collectively referred to as a Network Control Program, or NCP, which provides primitives for creation, control of data flow, and destruction of connections. An excellent survey of techniques used to implement the NCPs of various ARPANET Host systems has been given by Postel.[9]

Two major inferences regarding the desirable characteristics of Host operating system structure may be drawn from the ARPANET's evolution. First, the real-time event-driven nature of Host-IMP and Host-Host interaction requires some form of multiprogramming (i.e., multiple-process) capability in the Hosts. A Host system which supports terminal access to a network, for example, might utilize a process for each terminal. Each of these processes waits until a network message is received for terminal display or until a key-code is received for transmission to a remote system.

A second significant structural requirement of Host operating systems is a mechanism for communication

* This work was supported by the Advanced Research Projects Agency, Department of Defense, through Contract Number N00014-73-C-0221, administered by the Office of Naval Research.

between processes residing within a given Host ("local" processes) and processes residing in other Hosts on the network ("remote" processes). This inter-process communication (IPC) facility is achieved by the transmission of messages between Hosts according to an agreed-upon protocol. Implementation of network-wide IPC mechanisms (such as those embodied by the ARPANET Host-Host protocol) is greatly facilitated by the presence of internal mechanisms for IPC within each Host system.[10] Systems which lack these capabilities force the NCP to accept this responsibility; this is usually a fairly major implementation task when processes reside in different protected regions or address spaces.

Another important decision that must be made relates to the way in which the NCP is included in an operating system. The NCP may be embedded in the Host's file system, allowing network "connections" to be created and data transfers to occur in the same fashion that files are opened and read/written. This greatly facilitates the implementation of higher level protocols (e.g., file transfer) because it enables the standard file system primitives to be used for data transfer on network connections. Such an approach is practical when network protocol software is included as an integral part of the development of an operating system or when significant modification to an existing operating system may be tolerated. A disadvantage is that the file system must be modified when changes occur to low-level Host-Host protocols.

To simplify implementation and maintenance, it is desirable that an NCP run within a normal "user" job under the system. If system-wide IPC facilities are non-existent, this technique is feasible only when: (1) it is possible for the user job to usurp control on certain system calls which are issued by other user jobs, or (2) the scope of interaction with network resources is limited to a set of processes within a job, rather than globally available to all jobs. The first case makes assumptions about the protection structure which exists in the system, and is usually impractical when jobs occupy mutually exclusive protected memory spaces. The second case is feasible when the Host's sole function in the network is the management of resources which are allocated to the NCP "user" job; this technique might be used, for example, to provide access to a large data base. In both of the above cases it is necessary that the NCP implement an inter-process communication mechanism.

When a robust IPC facility is provided by an existing Host system it is possible to allow processes within the Host to communicate on a network-wide basis with minimal system modification. In this case, an NCP may exist as a user job, making use of the operating system's IPC facility to accept commands from the Host's processes (such as requests to open connections) and to handle the data transfer between local and remote processes.

Designers of Host operating systems for packet-switching networks must be sure that the chosen architecture provides sufficient flexibility. This is exemplified by the evolution of an ARPANET standard Host-Host protocol as well as special-purpose protocols for inter-network communication and packetized speech transmission. In some systems additional flexibility has been obtained by system calls which allow processes to intercept certain arriving messages and to transmit network messages directly, rather than forcing all network communication to occur by means of a standard Host-Host protocol.

HOST IPC MECHANISMS

Techniques for inter-process communication and synchronization in multiprogramming systems have received a good deal of attention.[6,11] There tend to be two strategies for implementing IPC systems. The first of these, like the telephone system, has required the establishment of a connection, or logical data path, before data may be transmitted between processes. In the distributed environment this strategy entails the utilization of special-purpose control messages which establish a name and control the data flow for the connection. The second approach has shunned the notion of prolonged connections, and performs the transfer of messages between processes whenever they mutually agree to communicate (e.g., process A requests to receive from process B while process B requests to send to process A). Control information (such as acknowledgment of messages received) is effectively embedded in each message exchanged between the Hosts. Walden has proposed such a connection-free mechanism[12] for inter-process communication within a packet-switching network. Metcalfe has also discussed the possibility of connection-free protocols.[13] In fact, it is becoming clear that the distinction should not actually be between connection-based protocols vs. connection-free protocols. The proper distinction to make is between protocols based on transmission of a single indefinitely long bit stream (starting when a connection is opened and ending when a connection is closed) vs. a stream of discrete messages (which may still require some connection-like control information). The message-based strategy is less sensitive to transmission errors which might occur in the communication subnetwork; the combination of control and data information within each transmitted message, for example, reduces the possibility of inconsistencies arising from the loss of a message. In addition, the message-based scheme requires minimal explicit connection setup, in some cases eliminating it entirely. Such an approach is being investigated in the design of Host-Host protocol techniques for interconnected packet-switching networks.[14]

There are various means by which IPC techniques have been implemented in operating systems. Some systems make use of a shared area of storage to pass messages between processes, utilizing the system's process synchronization techniques to announce the existence and acknowledgment of these messages. In other cases there exist special system calls (e.g., SEND, RECEIVE) which transfer data from a specified sender's buffer address to a matching receiver's buffer address. Akkoyunlu, Bernstein, and Schantz[15] have proposed a system (SBS) which supports inter-process communication and file I/O activities in a unified fashion. This approach has the distinct advantage of allowing processes to access data files stored within a remote system in the same way they would if the files were stored locally.

A similar IPC mechanism has been implemented in the ELF system[16] by means of the I/O primitives provided by the operating system. This technique enables the NCP to be included in the system as a user job, and all communication with the ARPANET occurs via this IPC structure. IPC occurs by means of a set of rendezvous-points, or ports, which appear identical to I/O devices except for differences in name. Processes may agree to communicate on a predesignated port, or may use a pair of such ports to exchange port numbers.

The ELF operating system provides a multiprogramming environment which allows creation and destruction of processes. Each process is named by means of a process-ID, owns an associated linked list structure called an event queue, and may be in one of two states: ready or blocked.

Processes synchronize by means of short (24-bit) messages which signal the occurrence of events. This is implemented by means of the following system primitives:

(1) SIGNAL (process-ID, event message)
(2) WAIT.

The SIGNAL primitive adds an event message to the event queue of process-ID; the SIGNALled process is placed in the ready state. The WAIT primitive tests the event queue of the active process, and places the process in the blocked state if the event queue is null; otherwise WAIT removes the first event message from the event queue and returns to the caller with the event message and the signalling process-ID. WAIT thus blocks the active process until an event message is placed on its event queue.

A process transmits data to another process by means of the primitive:

WRITE (port, mode, addr, count, event message),

in which port is the port name, mode denotes a stream or record-oriented transfer, addr is the address of data to be transmitted, count is the number of bytes for transfer, and event is an event message which the process wishes to receive (from WAIT) when the message has been sent. In record mode transfers the process is signalled when one or more bytes are taken by a receiver; in stream mode transfers the process is signalled when all bytes of the message are taken.

A process receives messages from another process by means of the primitive:

READ (port, mode, addr, count, event message).

All arguments are identical to those of the WRITE primitive. In record mode the receiving process is signalled when one or more bytes are placed in its input buffer; in stream mode the receiver is signalled only when all of the requested bytes have been placed in its input buffer. This allows a receiving process to reserve a large input buffer and wake-up when any data has been placed in its buffer (as is the case for a process awaiting input from the network and destined to be displayed on the user's terminal). Processes issuing WRITE or READ requests cause entries to be placed on a queue for the specified port; entries are removed from the queue when matching WRITEs and READs occur, and the appropriate transfer conditions (i.e., record or stream) are satisfied.

An additional primitive is provided to aid processes (e.g., and NCP) in gauging their allocation of buffer storage.

STAT (port)

returns the number of bytes which are queued to be written or read on the specified port. Special consideration is given when the requested count is specified as 0. In this case the WRITEing or READing process is signalled if there is a matching request on the specified port; the state of the port is unaffected in this case, and the process may then issue a normal WRITE or READ to transfer the pending data. This enables a process to wait for a matching request without locking up an input buffer for an unknown period of time.

OTHER HOST FACILITIES

Thus far we have dealt primarily with facilities for inter-process communication among Hosts in a distributed network. The capability of the Host as a viable network resource, however, depends heavily on services available in the Host operating system. In many cases this involves more than an implementation of protocols, and requires significant augmentation of Host facilities. Hosts providing interactive services, for example, must allow network access to occur in a fashion which is compatible with local terminal I/O. Many time-shared operating systems utilize a dedicated "logger" process which awaits the activation of previously dormant terminals. A mechanism is required to enable notification of the logger process when a network port becomes assigned on behalf of a user at a remote terminal. In a system which treats files and network connections uniformly this may be achieved by means of a system primitive which assigns a pair of files (i.e., the network connections) as controlling input and output data streams for the logger.

Batch-oriented service Hosts require the ability to redirect input and output files to network ports. This task is facilitated in systems which support remote job entry, allowing the set of network ports to be associated with a pseudo remote job entry terminal.

Host operating systems supporting file transfer must supply a flexible set of primitives for the allocation, updating, deletion, and directory maintenance of the file system.

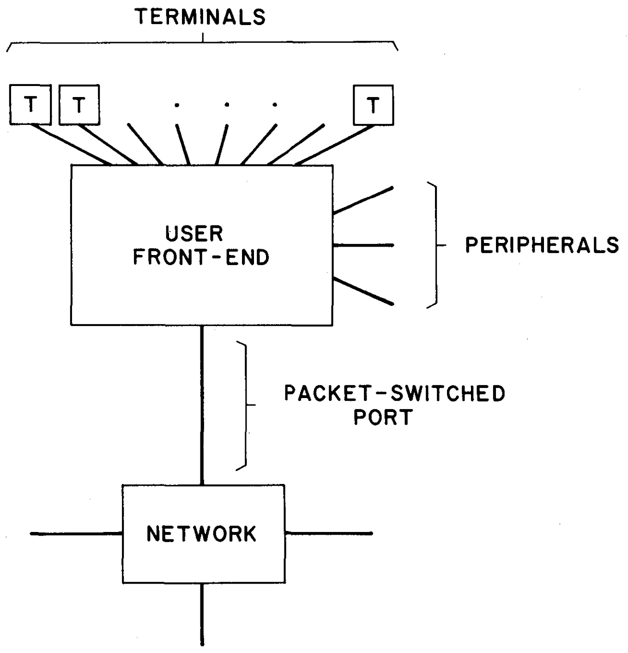

Figure 1—Typical user front-end system

In addition, a means of user authentication is usually required to provide an access control and accounting mechanism for the Host's resources.

FRONT-END SYSTEMS

In a number of cases it has been desirable to minimize the changes to a Host operating system when adding a pre-existing Host to the network. In other cases, it has been desirable for reasons of reliability, flexibility, or increased Host system performance, to clearly separate the network functions from the Host. These two cases have frequently been handled by the addition of a front-end system as an interface between the network and the Host. Front-end Hosts are usually implemented by means of small computers utilizing operating systems which support network protocols as well as various terminal and peripheral handlers.

The most common application of front-end systems in the ARPANET results from the need for user terminal and peripheral access to network computing resources. The TIP,[17] ANTS,[18] and ELF[16] systems are examples of such user front-ends, shown in Figure 1. In the above systems, for example, users at terminals may LOGIN at the front-end and use the Telnet protocol to connect to various server sites on the network. Data may be transferred to or from attached peripheral devices (line printers, magnetic tape units, disk drives) by means of a file transfer protocol.

A second type of front-end system facilitates the attachment of a computing resource (server Host) to a network. This approach aims to relieve the server Host of network communication tasks, such as those required to support Host-Host protocols, and is desirable when software modifications to an existing Host system are prohibited.

The structural requirements of the operating system in the server front-end are similar to those in the user front-end. (In fact, systems may serve both as a user and as a server front-end.) Figure 2 illustrates the means of interconnection of server and server front-end. Processes in the server front-end respond to (user) requests from the network, and provide access to the server via a number of server/front-end ports (hard-wired connections). For example, a front-end system might be connected to a number of terminal interface ports belonging to a server system which supports interactive terminal access. When a network message which requests connection to the server is received by the front-end, a process is created in the front-end; this process then allocates an unused server/front-end port and initiates requests for (full-duplex) data transfer between the remote network process and the server front-end port. This approach allows the

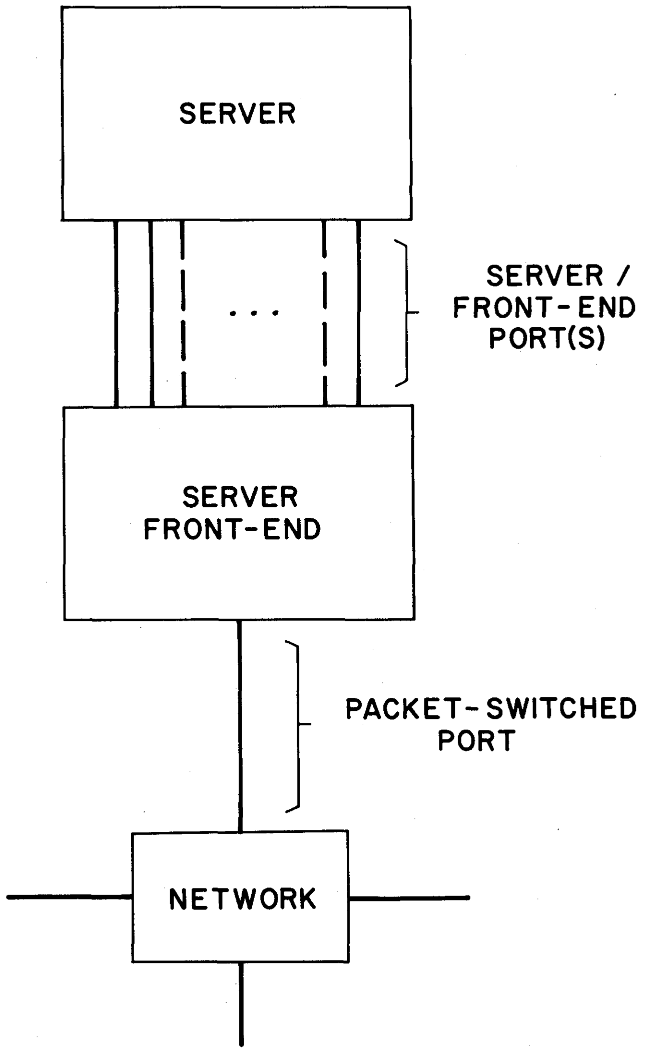

Figure 2—Server front-end system

server to be accessed from the network with no change to server software; however, it lacks generality, and server capabilities are limited to those functions associated with terminals. The server front-end model also applies to batch-oriented systems, in which case the style of connection between the server and the server front-end might resemble a set of card readers or printers.

When software additions are feasible in the server, a server/front-end protocol may be utilized to permit communication over a single hard-wired connection. As in the case of NCP implementations described earlier, however, general access to network resources by processes within the server requires some form of server IPC capability.

Interestingly, in some cases, functions may be performed by a large Host on behalf of a small front-end. For example, the means of providing control of access to a network requires access to a data base of user names, passwords, account numbers, and so forth. This may be accomplished by allowing the front-end system to simultaneously request connection (broadcast) to a number of "server" Hosts, accepting the first successful completion of a connection, and then requesting the attached server to perform the user authentication task; such a cooperative technique is in use between the TIP and TENEX RSEXEC[19] systems on the ARPANET. This type of interaction between Host systems is an example of automated resource sharing.

AUTOMATED RESOURCE SHARING

Host operating systems in the network environment may be structured to allow automatic utilization of resources in other Hosts. This subject has generated considerable interest because there is need for distributed data bases and load sharing in the network environment. As the size of network user communities expands, it becomes increasingly important to automate the allocation of processing and storage resources to allow their widespread and efficient use, while reducing problems faced by unsophisticated users. The realization of these techniques involves a diversion from the traditional image of centralized operating system structures; it involves the management of resources which are distributed, as opposed to centralized, by means of the coordination of distributed processes. This coordination requires within each centralized system a well-defined protocol and a flexible set of facilities to enable the processes to reliably carry out the management of network-wide resources.

Facilities provided by such a distributed operating system make possible common file naming schemes which are global to the network, thereby freeing a user of the responsibility of remembering a particular file's location. In addition, a network-wide user (directory) naming convention may be established. Tools for user-user communication, such as mail facilities, direct terminal intercommunication, and conferencing are needed on a network-wide basis. These capabilities will become increasingly desirable as the number of widely-dispersed service systems and users requiring access to shared data bases increases. Techniques are required for automatic archiving of network data bases; similarly, there is need for automatic retrieval of files (or portions thereof) upon reference, in much the same way as information flows between levels of storage in current centralized hierarchical storage systems. Of course, solutions to these problems raise a number of complex data management, accounting and security issues.

Development of the TENEX RSEXEC[19] System has been a step in this direction, providing users with a unified network file directory structure and terminal-terminal communication capabilities. The associated RSEXEC protocol allows the involvement of a number of TENEX sites, and allows other non-TENEX systems to participate. The success of this effort has been made possible in part by the characteristics of the TENEX operating system, which provides a tree-structured process and virtual memory capability.[20]

The development of distributed operating system structures requires a number of support facilities within each centralized system. A "system within a system" approach is desirable in which the resource-sharing processes utilize system primitives available in the parent operating system and create sub-processes which carry out remotely requested tasks. The creator processes intercept system calls issued by the sub-processes, providing them with a different virtual programming environment from that provided by the parent operating system. For example, a sub-process may access files in a network-wide directory structure by means of what it thinks are standard system calls.

Load-sharing techniques are aimed at an allocation of resources which provides distribution of load or assignment of processing tasks to the most appropriate server sites. For example, it is possible to allow tasks to be cooperatively carried out (simultaneously) by a number of servers; a protocol is being designed and implemented which allows processes to call procedures which execute on differing machines.[21] Basic problems are encountered in attempts at dynamic distribution of load: programs may depend on locally available data bases or on hardware or software peculiarities of a particular system.[22] These problems may be relieved by establishing a "standard" set of programs (e.g. editors, text-formatting programs, compilers) and conventions which guarantee compatibility with the virtual programming environment provided by the resource-sharing processes. The practicality of performing this type of load sharing, however, hinges on the existence of methods of accessing remotely distributed data.

SUMMARY

This paper has discussed structural characteristics which are required in Host operating systems for a packet-switching network; the need for flexible mechanisms for inter-process communication in the network Host (along with an example of an IPC mechanism for an ARPANET

Host System); the function and structure of "front-end" Host systems as network user and server interfaces; and finally, the need for development of automated techniques for managing resources in the distributed environment. These techniques will eventually provide capabilities for widespread access to shared data bases and new services for user-user communication. Their development requires well-structured centralized operating systems which serve as building blocks in the framework of a network-wide distributed operating system.

ACKNOWLEDGMENTS

The author wishes to thank R. Brooks, J. Malman, J. Miller, M. Retz, and D. Walden who assisted in the development of this manuscript.

REFERENCES

1. Roberts, L. G., F. D. Wessler, "Computer Network Development to Achieve Resource Sharing," *Proceedings of AFIPS SJCC,* 1970, pp. 543-549.
2. Kahn, R. E., "Resource Sharing Computer Communications Networks *Proceedings of the IEEE,*" Vol. 60, No. 11, November 1972, pp. 1397-1407.
3. Pouzin, L., "Presentation and Major Design Aspects of the CYCLADES Computer Network," *Proc. 3rd. Data Communications Symp.,* 1973, pp. 80-88.
4. Heart, F. E., R. E. Kahn, S. M. Ornstein, W. R. Crowther, D. C. Walden, "The Interface Message Processor for the ARPA Computer Network," *Proceedings of AFIPS SJCC,* 1970, Vol. 36, pp. 551-567.
5. Dijkstra, E. W., "The Structure of the "THE"—Multiprogramming System," *Communications of the ACM,* Vol. 11, No. 5, May 1968, pp. 341-346.
6. Horning, J. J., B. Randell, *"Process Structuring," ACM Computing Surveys,* Vol. 5, No. 2, March 1973, pp. 5-30.
7. McKenzie, A. A., *HOST/HOST Protocol for the ARPA Network,* National Technical Information Service, AD757680.
8. Crocker, S. D., R. M. Metcalfe, J. B. Postel, J. F. Heafner, "Function-Oriented Protocols for the ARPA Computer Network," *Proceedings of AFIPS SJCC,* 1972, Vol. 40, pp. 271-279.
9. Postel, J. B., *Survey of Network Control Programs in the ARPA Computer Network,* MITRE Technical Report #6722, October 1974, Revision 1, 89 pp.
10. Metcalfe, R. M., "Strategies for Operating Systems In Computer Networks," *Proceedings of the ACM Annual Conference,* August 1972, pp. 278-281.
11. Spier, M., E. Organick, "The MULTICS Interprocess Communication Facility," *Proceedings ACM Second Symposium on Operating System Principles,* Princeton University, October 20-22, 1969, pp. 83-91.
12. Walden, D. C., "A System for Interprocess Communication in a Resource Sharing Computer Network," *Communications of the ACM,* Vol. 15, No. 4, April 1972, pp. 221-230.
13. Metcalfe, R. M., *Packet Communication,* MIT Report, MAC TR-114, December 1973.
14. Cerf, V. G., R. E. Kahn, "A Protocol for Packet Network Intercommunication," *IEEE Transactions on Communications,* Vol. COM-22, No. 5, May 1974, pp. 637-648.
15. Akkoyunlu, E., A. Bernstein, R. Schantz, "Interprocess Communication Facilities for Network Operating Systems," *Computer,* June 1974, pp. 46-55.
16. Retz, D., J. Miller, J. McClurg, B. Schafer, *ELF System Programmer's Guide,* September 1974, Speech Communications Research Lab, Inc., Santa Barbara, California.
17. Ornstein, S. M., F. E. Heart, W. R. Crowther, H. K. Rising, S. B. Russell, A. Michel, "The Terminal IMP for the ARPA Computer Network," *Proceedings of AFIPS SJCC,* 1972, Vol. 40, pp. 243-254.
18. Bouknight, W. J., S. Denenberg, *ANTS—A New Approach to Accessing the ARPA Network,* University of Illinois Report CAC No. 47, July 1972.
19. Thomas, R. H., "A Resource Sharing Executive for the ARPANET," *Proceedings of AFIPS NCC,* 1973, pp. 155-163.
20. Bolt Beranek and Newman Inc., Computer Science Division, *TENEX JSYS MANUAL—A Manual of TENEX Monitor Calls.*
21. White, J. E., *Procedure Call Protocol Specification,* November 1974, Augmentation Research Center, Stanford Research Institute.
22. Dennis, J. B., "A Position Paper on Computing and Communications," *Communications of the ACM,* Vol. 11, No. 5, May 1968, pp. 370-377.