M.I.T. Laboratory for Computer Science
Computer Systems Research Division

June 30, 1977
Request for Comments No. 145

ANNUAL REPORT: July, 1976 -- June, 1977

by D.D. Clark, J.H. Saltzer, and L. Svobodova

During this year, the Computer Systems Research Division completed one major project, the information sharing kernel design project, and made significant progress on two others, the study of distributed systems and implementation of a local network. We also continued support of the ARPANET and NSW on Multics. These activities are described in the following sections.

I. THE INFORMATION SHARING KERNEL DESIGN PROJECT

This year we completed a three year project to carry out engineering studies whose goal was to demonstrate the feasibility of producing a full function general purpose operating system whose central supervisor code is simple enough that its correct operation can be certified by some form of auditing. The term "security kernel" is often used to describe this body of critical code, since the functions that must be included in this code are precisely those that insure the correct operation of the system, and insure the integrity of the information stored in the system. This engineering study was part of a larger project, the Guardian project, to produce a prototype of a certifiable operating system, based on the Multics system. The Guardian

project included development of models to characterize security in a computer system, development of formal specification techniques for operating systems, and actual implementation of a system matching the models.

The general strategy of this engineering study involved identifying all reasonable-sounding proposals for simplifying the Multics kernel, and selecting for trial implementation those that could not be accepted as obviously straightforward or rejected as obviously inappropriate. Three kinds of redesign proposals emerged: 1) removing from the kernel those formerly protected supervisor functions that did not really require that protection; 2) taking advantage whenever possible, of the natural separation afforded by processes in distinct address spaces communicating at arms length to implement protection functions; and 3) using more systematic program structuring techniques for implementing the remaining kernel functions, so that the result might be easier to verify.

Probably the most interesting and important result of this work is the invention of a file system and processor multiplexing organization that eliminates the complicating cycles of dependency normally found among the modules of an operating system kernel. The organization is based on the discipline of type extension, a strategy that has been used previously to organize application programs, but has heretofore not been applied to the structure of an operating system itself. Inside an operating system, careful analysis is required to identify all intermodule dependencies. The opportunity exists, for example, for an operating system module to produce dependency loops by participating in the implementation of its own execution environment. Such opportunities are less of a problem for application programs, which typically depend on the operating system to provide their

execution environment. Our study suggests that in a properly structured system, all dependencies that cannot be eliminated will fall into one of five categories, as follows. A module M is dependent on some other module if and only if:

- the other module manages some object that is a component of the object defined by M,
- that module provides a map used to relate names used by M to lower level objects,
- that module provides the containers for the algorithms and temporary storage for M,
- that module defines the address space in which M executes,
- that module implements the interpreter (the real or virtual processor) that executes the algorithms of M.

Using the rationale just described, and with the five kinds of dependencies in mind, it was possible to design a loop-free structure of object managers that implement the complete functionality required in the Multics kernel.

We summarize our experience in applying the type extension rationale to structuring the Multics kernel as follows. Most systems appear to have a loop-free dependency structure if viewed from far enough away. The obvious component relationships and the obvious operations follow loop-free paths among the modules. On close inspection, however, map, program, address space, and interpreter dependencies will almost certainly generate loops in the system designed without loop avoidance as a primary objective. The map, program and address space loops usually are broken easily (at least during the design stage) by introducing new object types to store the maps, programs, and address space definitions. The interpreter dependency loops appear to be

eliminated in most systems by using a two level implementation of processes. The most difficult and subtle structural problems are caused by exception handling - especially when the exceptions are part of the mechanisms that control resource usage. The difficulty is partly intrinsic -- such exceptions tend to occur at low levels in the system but be related to high level objects -- and partly methodological -- resource usage controls and the paths followed to deal with exceptions tend to be added to a design last.

It was our expectation that the structural simplifications to the kernel would be accompanied by a reduction in the size of the kernel, as measured in lines of source code. The size of the Multics kernel at the start of the project was 54,000 lines of source code, a bulk sufficiently staggering to inhibit any serious thought of conclusive auditing. Our application of the three design procedures mentioned above produced a version of the kernel approximately half the size of the original. And we expect further size reductions would be possible, were our proposals carried through to all areas of the kernel to which they would apply. An unresolved question is whether the kernel must enforce all security requirements, or only those related to some external standard such as the military model of non-discretionary levels and categories. Had our kernel enforced only the latter, it would have been somewhat smaller, though considerable work seems necessary to decide exactly how much smaller.

Experiments with components of the system that we rewrote indicate that the structural modifications we proposed did not have a significant performance impact on the system, and we conclude that a secure system need have no performance penalty. The most serious impact on performance in our work comes from the use of a high level language, and presumably this

difficulty could be minimized if a high level language were used that is easier to compile efficiently than full PL/I.

The primary conclusion of this project is that the kernel of a general purpose operating system can be made significantly simpler by imposing first clear criteria as to what should be in it -- the kernel concept --, and second a design discipline based on type extension. It is also apparent that minor adjustments of the underlying hardware architecture can make a significant difference in operating system complexity, and similarly that minor variations in the semantics of the user interface can make major differences in the complexity of implementation of the kernel.

II. RESEARCH PROBLEMS OF DECENTRALIZED SYSTEMS WITH LARGELY AUTONOMOUS NODES

A currently popular systems research project is to explore the possibilities and problems for computer system organization that arise from the rapidly falling cost of computing hardware. Interconnecting fleets of mini- or micro-computers and putting intelligence in terminals and concentrators to produce so-called "distributed systems" has recently been a booming development activity. While these efforts range from ingenious to misguided, many seem to miss a most important aspect of the revolution in hardware costs: that more than any other factor, the entry cost of acquiring and operating a free-standing, complete computer system has dropped and continues to drop rapidly. Where a decade ago the capital outlay required to install a computer system ranged from $150,000 up into the millions, today the low end of that range is below $15,000 and dropping.

The consequence of this particular observation for system structure comes from the next level of analysis. In most organizations, decisions to make

capital acquisitions tend to be more centralized for larger capital amounts, and less centralized for smaller capital amounts. On this basis we may conjecture that lower entry costs for computer systems will lead naturally to computer acquisition decisions being made at points lower in a management hierarchy. Further, because a lower-level organization usually has a smaller mission, those smaller-priced computers will tend to span a smaller range of applications, and in the limit of the argument will be dedicated to a single application. Finally, the organizational units that acquire these computers will by nature tend to operate somewhat independently and autonomously from one another, each following its own mission. From another viewpoint, administrative autonomy is really the driving force that leads to acquisition of a computer system that spans a smaller application range. According to this view, the large multiuser computer center is really an artifact of high entry cost, and does not represent the "natural" way for an organization to do its computing.

A trouble with this somewhat oversimplified analysis is that these conjectured autonomous, decentralized computer systems will need to communicate with one another. For example: the production department's output will be the inventory control department's input, and computer-generated reports of both departments must be submitted to higher management for computer analysis and exception display. Thus we can anticipate that the autonomous computer systems must be at least loosely coupled into a cooperating confederacy that represents the corporate information system. This scenario describes the corporate computing environment, but a similar scenario can be conjectured for the academic, government, military, or any other computing environment. The conjecture

described here is being explored for validity in an undergraduate thesis by Cecilia d'Oliveira.

The key consequence of this line of reasoning for computer system structure, then, is a technical problem: to provide coherence in communication among what will inevitably be administratively autonomous nodes of a computer network. Technically, autonomy appears as a force producing incoherence: one must assume that operating schedules, loading policy, level of concern for security, availability, and reliability, update level of hardware and software, and even choice of hardware and software systems will tend to vary from node to node with a minimum of central control. Further, individual nodes may for various reasons occasionally completely disconnect themselves from the confederacy, and operate in isolation for a while before reconnecting. Yet to the extent that agreement and cooperation are beneficial, there will be need for communication of signals, exchange of data, mutual assistance agreements, and a wide variety of other internode interaction. We hypothesize that one-at-a-time ad hoc arrangements will be inadequate, because of their potential large number and the programming cost in dealing with each node on a different basis.

Coherence can be sought in many forms. At one extreme, one might set a company-wide standard for the electrical levels used to drive point-to-point communication lines that interconnect nodes or that attach any node to a local communication network. At the opposite extreme, one might develop a data management protocol that allows any user of any node to believe that there is a central, unified database management system with no identifiable boundaries. The first extreme might be described as a very low-level protocol, the second

extreme as a very high-level protocol, and there seem to be many levels in between, not all strictly ordered.

By now, considerable experience has been gained in devising and using relatively low-level protocols, up to the point that one has an uninterpreted stream of bits flowing from one node of a network to another. The ARPANET and TELENET are perhaps the best-developed examples of protocols at this level, and local networks such as the ETHERNET and the Irvine Ring network provide a similar level of protocol on a smaller scale geographically. In each of those networks, standard protocols allow any two autonomous nodes (of possibly different design) to set up a data stream from one to the other; each node need implement only one protocol, no matter how many other differently designed nodes are attached to the network. However, standardized coherence stops there; generally each pair of communicating nodes must make some (typically ad hoc) arrangement as to the interpretation of the stream of bits: does it represent a stream of data, a set of instructions, a message to one individual, etc. For several special cases, such as exchange of mail or remotely submitting batch jobs, there have been developed higher-level protocols; there tends to be a distinct ad hoc higher-level protocol invented for each application. A Master's thesis by Paul Levine explored some of the problems of protocols that interpret and translate data across machines of different origin.

The image of a loose confederacy of cooperating autonomous nodes requires at a minimum the level of coherence provided by these networks; it is not yet clear how much more is appropriate, only that the opposite extreme in which

the physically separate nodes effectively lose their separate identity is excluded by the earlier arguments for autonomy. Between lies a broad range of possibilities that need to be explored.

Coherence and the object model

During the current year, members of the Computer Systems Research Division held a graduate-level seminar that explored this area of coherence among interconnected systems, and developed a framework for discussion that allows one to pose much more specific questions. The first conclusion of this work is that to put some structure on the range of possibilities, it is appropriate to think first in terms of familiar semantic models of computation, and then to inquire how the semantic model of the behavior of a single node might be usefully extended to account for interaction with other, autonomous nodes. To get a concrete starting point that is as developed as possible, we gave initial consideration to the object model*. Under that view, each node is a self-contained system with storage, a program interpreter that is programmed in a high-level object-oriented language such as CLU or Alphard, and an attachment to a data communication network of the kind previously discussed.

We immediately observed that several interesting problems are posed by the interaction between the object model and the hypothesis of autonomy. There are two basic alternative premises that one can start with in thinking about how to compute with an object that is represented at another node; send instructions about what to do with the object to the place it is stored, or

* Two other obvious candidates for starting points are the data flow model and the actor model, both of which already contain the notion of communications; since neither is developed quite as far as the object model we have left them for future examination.

send a copy of the representation of the object to the place that wants to compute with it. (In-between combinations are also possible, but conceptually it is simpler to think about the extreme cases first.) An initial reaction might be to begin by considering the number of bits that must be moved from one node to another to carry out the two alternatives, but that approach misses the most interesting issues: reliability, integrity, responsibility for protection of the object, and naming problems. Suppose the object stays in its original home. Semantics for requesting operations, and reporting results and failures are needed. For some kinds of objects, there may be operations that return references to other, related objects. Semantics to properly interpret these references are required. Checking of authorization to request operations is required. Some way must be found for the (autonomous) node to gracefully defer, queue, or refuse requests, if it is overloaded or not in operation at the moment.

Suppose on the other hand, that a copy of the object is moved to the node that wants to do the computation. Privacy, protection of the contents, integrity of the representation, and proper interpretation of names embedded in the object representation all are problems. Yet, making copies of data seems an essential part of achieving autonomy from nodes that contain needed information but aren't always accessible. Considering these two premises as alternatives seems to raise simultaneously so many issues of performance, integrity of the object representation, privacy of its content, what name is used for the object, and responsibility for the object, that the question is probably not posed properly. However, it begins to illustrate the range of considerations that should be thought about. We have identified the following more specific, problems that require solutions:

1. To arrange systematically that an object have multiple representations at one point in time but stored at different places. One would expect to achieve reliability and response speed this way. An example of non-systematic multiple representation occurs whenever one user of a time-sharing system confronts another with the complaint, "I thought you said you fixed that bug", and receives the response, "I did. You must have gotten an old copy of the program. What you have to do is type..." Semantics are needed to express the notion that for some purposes any of several representations are equally good, but for other purposes they aren't.

2. An object at one node needs to "contain" (for example, use as part of its representation) objects from other nodes. This idea focuses on the semantics of naming remote objects. It is not clear whether the names involved should be relatively high-level (e.g., character-string file names) or low-level (e.g., segment numbers).

3. Related to the previous problem are issues of object motion: suppose object A, which contains as a component object B, is either copied or moved from one node to another, either temporarily or permanently. Can object B be left behind or be in yet another node? The answer may depend on the exact combination of copy or new, temporary or permanent. Autonomy is deeply involved here, since one cannot rely on availability of the original node to resolve the name of B.

4. More generally, semantics are needed for gracefully coping with objects that aren't there when they are requested. (Information stored in autonomous nodes will often fall in this category.) This idea seems

closely related to the one of coping with objects that have multiple versions and the most recent version is inaccessible*.

5. Algorithms are needed that allow atomic update of two (or more) objects stored at different nodes, in the face of errors in communication and failures of individual nodes**. There are several forms of atomic update: there may be consistency constraints across two or more different objects (e.g., the sum of all the balances in a bank should always be zero) or there may be a requirement that several copies of an object be kept identical. The semantic view that objects are immutable may provide a more hospitable base for extension to interaction among autonomous nodes than the view that objects ultimately are implemented by cells that can contain different values at different times. (The more interesting algorithms for making coordinated changes in the face of errors seem to implement something resembling immutable objects).

Constraining the range of errors that must be tolerated seems to be a promising way to look at these last two problems. Not all failures are equally likely, and more important, some kinds of failures can perhaps be guarded against by specific remedies, rather than tolerated. For example, a common protocol problem in a network is that some node both crashes and restores service again before anyone notices; outstanding connections through

* Semantics for dealing systematically with errors and other surprises have not really been devised for monolithic, centralized systems either. However, it appears that in the decentralized case, the problem cannot so easily be avoided by the ad hoc tricks or finesse as it was in the past.

** Most published work on making atomic updates to several sites has concentrated on algorithms that perform well despite communication delay or that can be proven correct. Unfortunately, algorithms constructed without consideration of reliability and failure are not easily extended to cope with those additional considerations, so there seems to be no way to build on that work.

the network sometimes continue without realizing that the node's state has been reset. A change in the semantics of the host-net interface could locally eliminate this kind of failure instead of leaving it as a problem for higher level protocols.

The following oversimplified world view, to be taken by each node may offer a systematic way to think about multiply represented objects and atomic operations: there are two kinds of objects, mine and everyone else's. My node acts as a cache memory for objects belonging to others that I use, and everyone else acts as a backing store. These roles are simply reversed for my own objects. (One can quickly invent situations where this view breaks down, causing deadlocks or wrong answers, but the question is whether or not there are real world problems for which this view is adequate.)

Finally, it is apparent that one can get carried away with ingenious algorithms that handle all possible cases. An area requiring substantial investigation is real world applications. It may turn out that only a few of these issues arise often enough in practice to require systematic solutions. It may be possible, in many cases, to cope with distant objects quite successfully as special cases to be programmed one at a time.

Other problems in the semantics of coherence

Usual models of computation permit only "correct" results, with no provision for tolerating "acceptably close" answers. Sometimes provision is made to report that no result can be returned. In a loose confederacy of autonomous nodes, exactly correct results may be unattainable, but no answer at all is too restricting. For example, one might want a count of the current number of employees, and each department has that number stored in its computer. At the moment the question is asked, one department's computer is

down, and its count is inaccessible. But a copy of last month's count for that department is available elsewhere. An "almost right" answer utilizing last month's count for one department may well be close enough for the purpose the question was asked, but we have no semantics available for requesting or returning such answers. A more extreme example surrounds an attempt to determine the sum of all checking account balances in the United States, by interrogating every bank's computer. An exact result seems both unnecessary and unrealistic to obtain. A general solution to this problem seems to require a perspective from Artificial Intelligence, but particular solutions may be programmable if there were available semantics for detecting that one object is an out-of-date version of another, or that a requested but unavailable object has an out-of-date copy. It is not clear at what level these associations should be made.

Semantics are also needed to express constraints or partial contraints of time sequence. (e.g., "reservations are to be made in the order they are requested, except that two reservation requests arriving at different nodes within one minute may be processed out of order.") Note that the possibility of unreliable nodes or communications severely complicates this problem.

The semantics of autonomy are not clear. When can I disconnect my node from the network without disrupting my (or other) operations? How do I refuse to report information that I have in my node in a way that is not disruptive? If my node is overloaded, which requests coming from other nodes can be deferred without causing deadlock?

Heterogeneous and Homogeneous Systems

A question that we have repeatedly encountered is whether or not one should assume that the various autonomous nodes of a loosely coupled

confederacy of systems are identical either in hardware or in lower level software support. The assumption of autonomy and observations of the way the real world behaves both lead to a strong conclusion that one must be able to interconnect heterogeneous (that is, different) systems. Yet, to be systematic, some level of homogeneity is essential, and in addition the clarity that homogeneity provides in allowing one to see a single research problem at a time is very appealing.

We now believe that the proper approach to this issue lies in careful definition of node boundaries. We insist that every node present to every other node a common, homogeneous interface, whose definition we hope to specify. That interface may be a native interface, directly implemented by the node, or it may be simulated by interpretation, using the (presumably different) native facilities of the node. This approach allows one to work on the semantics of decentralized systems without the confusion of hetrogeneity, yet it permits at least some non-conforming systems to participate in a confederacy. There is, of course, no guarantee that an arbitrary previously existing computer system will be able to simulate the required interface easily or efficiently.

Conclusion

The various problems uncovered in the course of this work are by no means independent of one another, although each seems to have a flavor of its own. In addition, they probably do not span the complete range of issues that should be explored in establishing an appropriate semantics for expressing computations in a confederacy of loosely coupled, autonomous computer systems. Further, some are recognizable as problems of semantics of centralized systems

that were never solved very well. But they do seem to represent a starting point that we expect to lead to more carefully framed questions and eventually some new conceptual insight.

III. A LOCAL NETWORK FOR LCS

During the year, development of the Local Network for the Laboratory for Computer Science progressed to the point where the first three nodes on the net are expected to be operational within the next two months. As discussed in detail in the sections below, the critical decisions concerning the hardware and protocols to be used on our network have been made during the last twelve months, making it possible for a variety of projects related to the network to proceed forward in parallel.

Hardware

As reported in the last annual report, our choices for the transmission technology to be used in the network quickly narrowed to two architectures: the ethernet developed by Boggs and Metcalfe at Xerox Palo Alto Research Center, and the ring network developed by Farber at the University of California, Irvine. The architecture and hardware of the ring network and the ethernet are very different, and, at first glance, the functional capabilities of the two seem quite different as well. However, discussions with Metcalfe and Farber, and with others in our laboratory, led to the conclusion that there are few inherent differences in the functional capabilities of the basic ethernet and ring network communications schemes. This made the choice between them a very difficult one. It appeared, in fact, that the important differences between the two networks were operational differences such as reliability, cost, and convenience, which could only be evaluated by comparing a running version of each network in a similar environment.

A way out of this dilemma was suggested when we discovered that we could design a network interface that, with minor modification, could operate either a ringnet or an ethernet. Thus, without procuring two complete sets of interface hardware, we can bring up both versions of the network and compare them operationally. Given this observation, we determined that we would construct the LCS Net in two subcomponents, one a ringnet and one an ethernet, and perform an operational comparison of the two. We have done some preliminary comparative analysis of the two.

The primary hardware component of our network is the Local Net Interface (LNI), which provides the means of connecting the various hosts to the network. The LNIs that we intend to use for the network have been designed by David Farber at the University of California, Irvine; they are a second generation ring interface that Farber is developing under contract with ARPA, based on the ring developed for the Irvine Distributed Computing System. We have been assisting in the design of these interfaces, so that we will be able to produce a version of this hardware that can drive an ethernet as well as a ringnet.

The LNI, as delivered by Farber, includes an interface to the PDP/11 Unibus. One of the tasks yet to be completed is the fabrication of an interface to connect the LNI to the PDP-10s in the building. It is possible tnat Farber will complete the design of a PDP-10 interface to the LNI; as an interim interface it appears very easy to attach the LNI to the TTL bus that is locally used for conenction to the PDP-10s. Eventually, the LNI will probably require a connection to the PDP-10s that runs at a higher speed than the TTL bus will permit.

A hardware project that was partially completed during the year is the interconnection of a microprocessor to the LNI. A microprocessor directly connectable to the network can be used in a variety of ways, for example as a controller for a computer terminal or other remote input/output device. The microprocessor selected for this first implementation was the Motorola M6800. The first applciation for the microprocessor will be as a terminal interface for the local network.

One of the important functions of our local network will be to provide a means of access to the ARPANET from the various machines at the laboratory. The interconnection between the local net and the ARPANET will be made using a PDP 11/35 that was provided for the project by ARPA. This machine will be used to perform the various protocol translations that will be required as part of the interconnection of the local network and the ARPA network. One project being performed at the laboratory is the development of a hardware interface to connect this PDP/11 to the ARPANET. The DEC interface is bulky, expensive, and not rapidly obtainable. We hope our local version will perform better on these counts.

Protocols

As part of the development of our local network, it was necessary for us to develop or select a low level protocol for end-to-end communication over the network. We chose as a starting point the Transmission Control Protocol, or TCP, but we permitted ourselves the option of changing the protocol slightly to better conform to our local needs as we saw them. The resulting protocol is called Data Stream Protocol, or DSP. DSP provides functionality equivalent to TCP, but is simpler, primarily due to the elimination of certain control functions and synchronizing algorithms.

We are currently involved in an effort to bring DSP and TCP together again, since TCP is the ARPANET standard for end-to-end communication in the "internet" environment. We have attended several meetings of the TCP working group, and have met with some success in our attempt to include in TCP some of the features in DSP.

DSP must be implemented on all the machines which we propose to connect to the local network. Our initial effort has been devoted to an implementation of DSP for the UNIX operating system on the PDP/11. One of the first machines to be connected to our local network will be the UNIX system in the Domain Specific Systems research group. In addition, the PDP/11 gateway to the ARPANET will run the UNIX operating system. An implementation of DSP (or perhaps TCP) is scheduled for the Multics system later in the calendar year. Preliminary plans have been made for implementation of DSP on the ITS machines, and we are considering how DSP might be implemented on the TENEX operating system. As part of the microprocessor project mentioned above, we have also implemented DSP for the M6800. The initial implementation on the M6800 required 1300 bytes of program, and although this size will undoubtedly increase as the implementation is polished, the size of the algorithm suggests that we were somewhat successful in our ambition that DSP be a fairly simple protocol.

Initially, the local net will use the same high level protocols that are now used in the ARPANET. It appears that the ARPANET protocols for remote login (TELNET), file transfer, and mail sending can be made to operate on top of DSP without major modification. Therefore, for systems that currently have software for conenction to the ARPANET, the only coding required as part of the interconnection to the local net is the implementation of DSP, and minor modification of existing higher level protocols. ARPANET software already

exists for all the machines currently scheduled for connection to the local network.

We have begun the design of higher level protocols to provide new services that seem appropriate in the local net. In particular, we have proposed a rather flexible scheme for naming and initiating connections to services in the local network. Examples of services that might be named using this mechanism are the delivery of a message to a specified mailbox, the updating of a file, or the remote login to a system. The mechanism uses decentralized active agents to provide an environment that is robust in the face of system failures. The names used are tree structured in order to deal in the natural way with name conflicts and to allow the easy definition of new services in a given context.

All of the network architectures that we have considered are completely insecure, since all messages being sent appear on all portions of the network. While our laboratory is a "benign" environement in which the needs for security of data communication are rather small, considerations of personal privacy continue to be relevant in an environment such as ours, so our needs for security, while minimal, are not zero. Also, we would like to design a network whose applicability extends to situations with stronger protection requirements than we have. For these reasons, we have studied the securing of information flowing through our local network by means of data encryption. Data encryption is becoming a viable possibility for a network even as simple as the one we contemplate here, because data encryption algorithms can now be obtained on a single chip. We have proposed a end-to-end encryption strategy using the NBS data encryption standard integrated into a modified version of DSP, which is essentially invisible to the higher level protocols. Its use in the local network could be made automatic, invisible and inexpensive. We feel

that the integration of some security mechanism into our network will considerably enhance the impact of our work in the outside world.

IV. ARPANET AND NSW SUPPORT

During the year, our group significantly reduced the level of effort committed to maintaining the ARPANET connection to the Multics system. Although Honeywell has not officially accepted support for the ARPANET software, it has agreed that it will attempt to modify the ARPANET software when necessary as a result of changes to other parts of the system. Therefore, we are somewhat relieved of the continued effort which has been required just to maintain the ARPANET in a stable condition. The only modifications to the software that we are performing at this point are changes required to support other research activities of our group.

We continue to improve the software implementing the higher level protocols on Multics, especially the programs for sending and receiving network mail. The Information Processing Center is currently providing computer time on Multics in support of our project to produce an installable program for reading and managing mail. We are also in the process of transferring to IPC the cost of managing the system services related to receiving and sending network mail.

A significant amount of effort has been invested in making Multics a participating member of the National Software Works. At this point, Multics is a legitimate tool-bearing host in the NSW. We are in the process of transferring continued support of NSW on Multics to the Rome Air Development Center.

Publications

Saltzer, J.H., "Technical Possibilities and Problems in Protecting Data in Computer Systems," in R. Dierstein, H. Fielder, and A. Schulz, Datenschutz und Datensicherung, J. P. Bachem Verlag, Cologne, Germany, September, 1976, pp. 27-36.

Gifford, D., "Hardware Estimation of a Process's Primary Memory Requirements," to be published in Comm. ACM in September, 1977.

Schroeder, M.D., Clark, D.D., and Saltzer, J.H., "The Multics Kernel Design Project," to appear in the Sixth ACM Symposium on Operating Systems Principles, November, 1977.

Reed, D.P., and Kanodia R.J., "Synchronization with Eventcounts and Sequencers," to appear in the Sixth ACM Symposium on Operating Systems Principles, November, 1977.

Svobodova, L., "Software Performance Monitors: Design Trade-Offs," Seventh International Conf. of the Computer Measurement Group, Atlanta, Georgia, November, 1977.

Kent, S., "Encryption-Based Protection for Interactive User-Computer Communication," to be presented at the ACM Fifth Data Communications Symposium, Snowbird, Utah, September, 1977.

Montgomery, W., "Measurements of Sharing in Multics," to appear in the Sixth ACM Symposium on Operating Systems Principles, November, 1977.

Other Reports

Clark, D.D., editor, "Ancillary Reports: Kernel Design Project," June, 1977, Laboratory for Computer Science Technical Memo TM-87.

Schroeder, M.D., Clark, D.D., Saltzer, J.H., and Wells, D.M., "Final Report of the Multics Kernel Design Project," June, 1977, submitted to Honeywell Information Systems Inc.

Theses Completed

Wajda, J.P., "A Methodology to Study Computer Language Performance," B.S. thesis, Department of Electrical Engineering and Computer Science, M.I.T., August, 1976.

Janson, P., "Using Type Extension to Organize Virtual Memory Mechanisms," Ph.D. thesis, Department of Electrical Engineering and Computer Science, M.I.T., September, 1976, also Laboratory for Computer Science Technical Memo TR-167.

Benjamin, A., "Improving Information Storage Reliability Using a Data Network," M.S. thesis, Department of Electrical Engineering and Computer Science, M.I.T., October, 1976, also Laboratory for Computer Science Technical Memo TM-78.

Hunt, D., "A Case Study of Intermodule Dependencies in a Virtual Memroy Subsystem," M.S. thesis, Department of Electrical Engineering and Computer Science, M.I.T., December, 1976, also Laboratory for Computer Science Technical Report TR-174.

Skalka, S.L., "Analysis of Simulation Models of a Multiprogrammed Demand Paging System," B.S. thesis, Department of Electrical Engineering and Computer Science, M.I.T., December, 1976.

Frydman, U., "Minicomputer Systems in the Automated Factory," B.S. thesis, Department of Electrical Engineering and Computer Science, M.I.T., January, 1977.

Goldberg, H.J., "A Robust Environment for Program Development," M.S. thesis, Department of Electrical Engineering and Computer Science, M.I.T., February, 1977, also Laboratory for Computer Science Technical Report TR-175.

Karger, P., "Non-Discretionary Access Control for Decentralized Computing Systems," M.S. thesis, Department of Electrical Engineering and Computer Science, M.I.T., May, 1977, also Laboratory for Computer Science Technical Report TR-179.

Luniewski, A., "A Simple and Flexible System Initialization Mechanism," M.S. thesis, Department of Electrical Engineering and Computer Science, M.I.T., May, 1977, also Laboratory for Computer Science Technical Report TR-180.

Mason, A., "A Layered Virtual Memory Manager," M.S. thesis, Department of Electrical Engineering and Computer Science, M.I.T., May, 1977, also Laboratory for Computer Science Technical Report TR-177.

Rodriguez, H., "Measuring User Characteristics on the Multics System," B.S. thesis, Department of Electrical Engineering and Computer Science, M.I.T., May, 1977.

Harriman, E.S., "A Microprocessor Based Implementation of a Data Stream Protocol Processor," B.S. Thesis, Department of Electrical Engineering and Computer Science, M.I.T., June, 1977.

Theses in Progress

Krizan, B., "A Minicomputer Network Simulation System," M.S. thesis, Department of Electrical Engineering and Computer Science, M.I.T., expected date of completion, July, 1977.

Ciccarelli, E., "Multiplexed Communication for Secure Operating Systems," M.S. thesis, Department of Electrical Engineering and Computer Science, M.I.T., expected date of completion, August, 1977.

d'Oliveira, C., "A Conjecture About Computer Decentralization," B.S. thesis, Department of Electrical Engineering and Computer Science, M.I.T., expected date of completion, August, 1977.

Talks and Presentations

Saltzer, J.H., "Pragmatic Approaches to Obtaining Correct Operating Systems"
given at: IBM Research Laboratory, Zurich, Switzerland, September, 1976
Cambridge Univeristy, England, September, 1976
Rutgers University, New Jersey, November, 1976
Industrieanlagen-Betriebsgesellschaft mbH, Munich, Germany, January, 1977
Central Computer Agency, London, England, January, 1977

"Decentralized Systems with Largely Autonomous Nodes"
given at: University of Waterloo, Toronto, Canada, June, 1977

Svobodova, L., "Distribution and Coherence in Computer Systems"
given at: McGill Univeristy, Montreal, Canada, June, 1977

"Computer Performance Measurement: Methods and Tools"
given at: Digital Equipment Corporation, Maynard, Mass., May, 1977

Clark, D.D., "A High-Speed Local Computer Network"
given at: IEEE Boston Chapter Communications Group, May, 1977

Wells, D., "The Multics Implementation of the National Software Works"
given at: RADC, Griffiss Air Force Base, New York, June, 1977

Reed, D., "Service Addressing Protocols for Local Networks"
given at: Bolt Beranek and Newman, Cambridge, Mass., March, 1977

Forsdick, H., "The Design of a Distributed Data Base Management System"
given at: Sperry Research Center, Sudbury, Mass., November, 1976

Kent, S.T., "End-to-End Communication Security Measures"
given at: System Development Corporation, Santa Monica, Calif., July, 1976
Information Sciences Institute, Univ. of Southern California, July, 1976
Xerox Palo Alto Research Center, California, August, 1976
National Bureau of Standards workshop, Gaithersburg, Maryland, Sept., 1976
Federal Telecommunications Standards Committee, Gaithersburg, Maryland, February, 1977
Sperry Univac, Roseville, Minnesota, March, 1977
GTE Sylvania, Neeham, Mass., May, 1977

Hunt, D., "A Case Study of Intermodule Dependencies in a Virtual Memory Subystem"
given at: Stanford Research Insitute, Menlo Park, Calif., December, 1976
Honeywell Systems and Research Center, Roseville, Minnesota, January, 1977.

Pogran, K., "The Evolution of the Multics System"
given at: University of California, Irvine, California, January, 1977

Committee Memberships

Saltzer, J.H.,	ARPA IPTO Security Working Group
Wells, D.,	ARPA IPTO NSW working group
Reed, D.,	ARPA IPTO TCP Working Group
Clark, D.D.	ARPA IPTO TCP Working Group

PERSONNEL, July, 1976 -- June, 1977

Faculty and Research Associates

David D. Clark
Fernando J. Corbató
David D. Redell
Jerome H. Saltzer (Division Head)
Michael D. Schroeder
Liba Svobodova

Professional Staff

Nancy C. Federman
Rajendra K. Kanodia
Robert F. Mabee
Kenneth T. Pogran
Douglas M. Wells

Support Staff

Virginia M. Newcomb
Muriel Webber

Undergraduate Students

Charles R. Davis
Cecilia R. d'Oliveira
Edward S. Harriman
Roy P. Planalp
Humberto Rodriguez, Jr.
Steven A. Swernofsky

Graduate Students

Arthur J. Benjamin
Eugene C. Ciccarelli
Harry C. Forsdick
Robert M. Frankston
Harold J. Goldberg
Andrew R. Huber
Douglas H. Hunt
Philippe A. Janson
Paul A. Karger
Stephen T. Kent
Allen W. Luniewski
Andrew H. Mason
Warren A. Montgomery
David P. Reed
Karen R. Sollins