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

January 13, 1977
Request for Comments No. 133

Multiplexed Communication for Secure Operating Systems

Eugene C. Ciccarelli

Attached is my recently accepted Master's thesis proposal.

Massachusetts Institute of Technology

Laboratory for Computer Science

Computer Systems Research Division

Proposal for Thesis Research in Partial Fulfillment

of the Requirements for the Degree of

Master of Science

Title: Multiplexed Communication for Secure Operating Systems

Submitted by: Eugene C. Ciccarelli

Signature of Author

550 Memorial Dr. Apt. 22D

Cambridge, Massachusetts 02139

Date of Submission: January 13, 1977

Expected Date of Completion: August, 1977

Brief Statement of the Problem:

In computer systems, network control programs are usually large and complex. As networks are added to the system, the size of the supervisor grows quickly because these control programs are implemented independently of each other; this rapid growth makes detailed understanding of the supervisor very difficult. The proposed thesis will present a design that removes most network-dependent code from the supervisor, leaving a very simple multiplexing mechanism largely shared between networks.

Supervisor Agreement:

The program outlined in this proposal is adequate for a Master of Science thesis. The supplies and facilities required are available, and I am willing to supervise the research and evaluate the thesis report.

David D. Clark, Research Associate in
Electrical Engineering and Computer Science

Introduction

My thesis will attempt to simplify the design of the multiplexed communication system within the supervisor of a general time-sharing computer. It is part of a larger project seeking to identify and design a security kernel for Multics. A major requirement of a security kernel is that its subsystems be small and simple, in order that they be well understood; currently, however, I/O subsystems are neither small nor simple. In Multics most of the supervisor's I/O subsystem deals with multiplexed communication; it therefore is a major obstacle to the design of a complete security kernel.

(The Multics system will be used as an example throughout this proposal and my thesis.)

In Multics there are two parts of the supervisor that handle multiplexed communication: one that communicates with a Front End Processor, to which user terminals and other devices may be attached; and one that communicates with the Arpanet. Using Arpanet terminology, I will call such modules "network control programs", or "NCP's". These two modules are large and complex, with almost no sharing between them. The strategy of separate and complete NCP's, each entirely within the supervisor drastically increases the amount of supervisor code as more networks are connected and more NCP's are implemented; the effort required to understand and ensure proper operation of the supervisor increases proportionally.

Thus, two goals of my design will be: reduce the bulk of the NCP supervisor code, and reduce the number of network dependencies within the supervisor -- allowing a uniform design that grows slowly as the number of NCP's increases.

Secure Multiplexed Communication

My thesis will model the multiplexed communication facility, and then proceed to examine functions in this model to determine which affect security and therefore must reside in the kernel. Before this can be done, security requirements must be explicit.

Security Requirements

Since communication I/O deals with entities outside the system being secured, the security requirements for I/O may not be obvious. If communication is to an unsecured host, data transferred to and from that host is not secure; however, we would like to ensure that such communication may occur without endangering the security of users who are not communicating with that unsecured host. Further, when two secured hosts are communicating via a secured network, the combination must be secure: the communication path must meet the security requirements of a path completely within one secured host. Precise details of I/O security requirements will be presented in my thesis.

Communication System Model

Diagram 1 shows a data flow model for a communication facility. Higher level (protocol) functions are on the left; basic multiplexing and finally the hardware interfaces are on the right. Currently, both the Front End Processor and Arpanet NCP's implement something like this model separately within the supervisor.

In my thesis I will use this model to identify attributes that cause functions to be placed in the supervisor and to show how the principles of separating policy from mechanism

and minimization of supervisor mechanism can be applied at the same time network dependencies are identified and separated. Most network dependencies can be mapped onto policy desicions about network-independent mechanisms.

Messages read from the network are stored in a multiplexed input buffer (MIB). The demultiplexing function (DMPX) examines each incoming message, checks its validity, and decides for whom it is intended; the message is put in a demultiplexed input buffer (DMIB)for its recipient. The recipient (e.g. user) process may then, at its leisure, call to read from the DMIB.

A sender process calls the WRITE function to send data; this is put in an demultiplexed output buffer (DMOB), which is marked ready for outputting. The multiplexing function (MPX) schedules the ready DMOB's and copies messages into the multiplexed output buffer (MOB). The multiplexor ensures that the from-name field in the message is correct.

Some of the demultiplexed streams carry protocol-specified control information. The PROTOCOL function shown communicates with local users and with protocol modules on other hosts to control the connections between users.

I will divide this model into two parts: basic multiplexing and protocol functions. My thesis will argue that while the former is a kernel function, the latter is not.

Basic Multiplexing

Since basic multiplexing will reside in the kernel, it must have a simple design. There are several problems that have to be faced, among them:

Network-dependent functions have to identified, minimized, and isolated from network-independent parts as much as possible. Since these functions constitute the part of the

kernel that grows when new networks are attached, minimizing their functionality lessens the extra effort needed to ensure that they are secure. Isolating them from network-independent functions simplifies their interactions with the rest of the kernel, and thus allows the kernel to be incrementally secured as networks are added.

The decision about what to do with incoming messages with header errors can complicate the demultiplexor mechanism. This issue is related to the previous, since this decision may be network dependent.

The DMIB's and DMOB's run into the small-object problem. It may be too expensive to keep these buffers in separate segments outside the supervisor; in addition, there may be naming constraints that limit the supervisor's ability to access many separate user segments. These buffers are then likely to be within shared, supervisor segments, and buffer management should be done in a simple network-independent manner if possible.

Each network's MIB and MOB will be expensive wired resources. Management of these buffers also is critical to keeping the kernel simple. Network dependence should be placed in the functions that access these buffers, not in the buffers themselves.

Principles of Design

In designing the basic multiplexing facility, I will use the following guidelines:

1. Isolate policy from mechanism.[Wulf 74]

2. Isolate and minimize network-dependent parts.

3. Use type extension to structure the various resources.[Janson 76] In particular, type extension can be used to separate network dependencies that might exist in the multiplexed buffers, building them upon network-dependent multiplexed-buffer types.

4. Mechanisms should be stateless as much as possible, so that understanding them

does not require knowing history of the multiplexing.

5. The multiplexing facility should consider each connection to be half-duplex; i.e., input mechanisms and states should not interact with output mechanisms and states.

Protocol Functions

Logically, per-connection protocol could be handled by the user domain, while any multiple-connection messages in general must be handled by a supervisor function. I will attempt to move as much per-connection protocol as possible out of the supervisor, removing most network dependence, complexity, and bulk.

Three important protocol issues can force code into the supervisor: simulation of parallel processes to handle network connections, facilities for interrupting a user computation, and ensuring that some incoming messages get handled very quickly. Each of these must be examined carefully to see when they are necessary, and what minimum supervisor mechanism is necessary to implement them.

"Parallel Processes"

The Multics Arpanet NCP implementation centers the protocol code around one "socket state" for each connection, and uses this state to control simulation of parallel processes to handle each connection. The assumption in the design is that the user cannot have (or it is too expensive to have in general) a separate process for this purpose. Whenever any process calls into the NCP it will cause the supervisor to handle any socket control that needs to be done. Thus the decision to simulate processes causes functions to be placed in the

supervisor and must be examined carefully.

Some functions don't need parallel handling, i.e. protocol exchanges that start and proceed synchronously, such as connection setup; they need no help from the supervisor. Other functions, such as connection termination, may start at any time, but then proceed synchronously. These functions need only a supervisor mechanism to start them, i.e. an interrupt facility. The user domain can do its parallel-process simulation in this manner without affecting security.

Interrupts

An interrupt facility is fundamental to multiplexed communication, and needs a kernel mechanism to handle it. However, this can be provided by a very simple, network-independent, stateless mechanism, e.g. a bit in an incoming message. Having such a mechanism within the kernel leaves the interpretation of the interrupt, a protocol issue, out of the supervisor.

If users can have a separate, parallel process synchronously reading incoming control messages, then this interrupt mechanism is not needed within the NCP. (It is provided by other inter-process signaling kernel mechanisms.)

Quick Response

The problem of very quickly responding to an incoming message can seriously complicate attempts to extract network protocol from the supervisor. This problem arises with respect to flow control: the issue is maintaining high throughput to a recipient that cannot buffer very much and yet can process its input at a high rate (but not necessarily at a

constant rate predictable by the sender). An example of this would be a display terminal attached directly to a high speed, local network with very little buffering except that in the hardware interface. If there is enough buffering at the receiver, the sending user process has enough time when it gets a flow control message to wake up and respond by issuing more output, keeping the overall bandwidth high. However, if there is not enough receiver buffering then the response of the user process will not be fast enough to maintain the desired bandwidth. This problem has, in the past, forced some flow control code not only into the supervisor, but into interrupt-time code.

There may be several ways of dealing with this problem:

First, it might be considered not to be a problem; since memory is getting much cheaper, it may be acceptable to require that enough buffering be used.

Second, if sender and receiver can agree on a method of predicting the receiver's processing rate as a function of the data, the kernel could provide some simple, network-independent mechanism for regulating the output rate, possibly by a time-stamp on messages (relative to the last message's output time or absolute). This would be a more effective way, for example, of handling known delays that occur when sending certain control-sequences to terminals (e.g. clear-screen, home-up) than inserting padding characters.

Third, the needed buffering and response may be provided in a separate processor, one in which a process can wake to handle frequent flow-control messages without loss of bandwidth. The user would send large, infrequent messages to the buffer-processor, which would break them into small, frequent messages to be sent to the receiver.

A buffer-processor could be associated closely with the writer's host, or it could be a separate host on the network. In either case the question arises: how much of this processor has to be secured? The answer depends upon the exact form of the security requirements for the I/O system, but consider a secure host sending via a secure network to

a secure receiver, with the information routed through an unsecured buffer-host; the resulting path may not be considered secure. There are two approaches to this problem: First, it may be possible to identify a very small kernel for this processor. Second, a low-level flow-control protocol could be embedded within a higher-level protocol using end-to-end encryption.[Kent 76] Encryption can provide the security instead of the buffer's kernel, and the buffer may be left unsecured. The encryption protocols, unlike flow-control protocols, do not demand quick response, and therefore may be handled by the end user processes without trouble. Note that both approaches benefit from a very simple flow-control protocol within higher-level protocols, and that the flow-control protocol can be under the control of the user processes, just as is the higher-level protocols.

Proposed Plan of Research

My research will consist of two parts: a comprehensive design of the secure multiplexed communication I/O facility, and a partial implementation to demonstrate the feasibility of my design.

The kernel design will stress network independence and minimum mechanism, leaving (as much as possible) network dependence and policy outside the kernel. The design will also include discussions of non-kernel facilities, since the thesis must show that removing much of currently supervisory functions can be done efficiently. Particular attention will be paid to the problems of high-bandwidth flow control under quick-response, small-allocation conditions. various solutions will be offered, dealing with fast processes, special processors, and buffer-hosts. Various protocols and their possible implementations will be discussed, including TCP, Arpanet Host-Host, Telenet, and DSP.[Cerf 76, ARPA 76, Telenet 75, DSP 76]

The specifics of the partial implementation will be worked out as the design progresses. The implementation will concentrate on showing the viability of the major ideas: it must show that a protocol can be implemented efficiently outside the supervisor, using network-independent kernel mechanisms. It will not offer several complete protocols, but will demonstrate a few different skeleton protocols showing differing approaches to issues such as flow control, interrupts, and re-use of sockets. In particular, since my research group is committed to implementing TCP in 1977, my implementation may be very closely tied with the group's TCP effort.

References

[ARPA 76] *Arpanet Protocol Handbook*, pp. 7-40, NIC# 7104, ARPA Netowork Information Center, Stanford Research Institute, April 1976.

[Cerf 74] Cerf, Vinton, et al., "Specification of Internet Transmission Control Program", INWG Note No. 72, December 1974 (Revised).

[Clark 74] Clark, David D., "An Input/Output Architecture for Virtual Memory Computer Systems", MAC TR-117, January 1974.

[DSP 76] "Protocols for the LCS Network", Internal LCS document, 1976.

[Janson 76] Janson, Philippe A., "Using Type Extension to Organize Virtual Memory Mechanisms", LCS TR-167, September 1976.

[Kent 76] Kent, Stephen T., "Encryption-Based Protection Protocols for Interactive User-Computer Communication", LCS TR-162, May 1976.

[Telenet 75] Telenet Communications Corporation, "System Planner's Guide for Host Computer Systems", March 1975.

[Wulf 74] Wulf, W., et al., "Hydra: The Kernel of a Multiprocessor Operating System", *CACM*, Vol. 17, No. 6, June 1974, pp. 337-345.

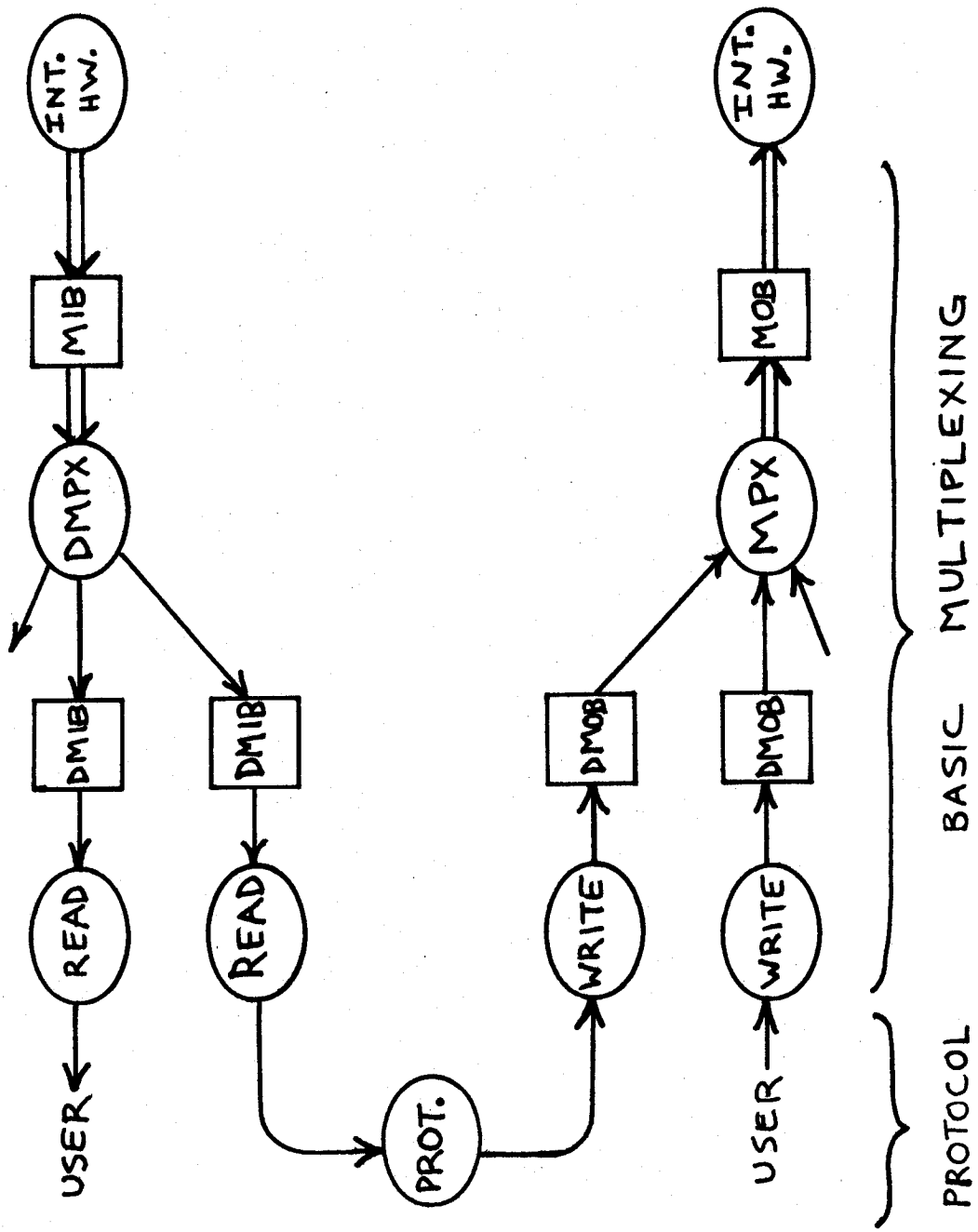

DIAGRAM 1: COMMUNICATION SYSTEM MODEL