



Laboratory for Computer Science

M. L. Dertouzos, Director

(formerly Project MAC)  
545 Technology Square  
Cambridge, Massachusetts 02139  
(617) 253-6016

J. Moses, Associate Director

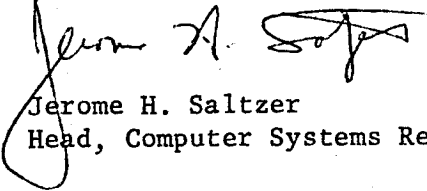
July 15, 1976

Mr. R. M. Carlson, Contracts Administrator  
Honeywell Information Systems Inc.  
7900 Westpark Drive  
McLean, Virginia 22101

Re: SDRL0004, M.I.T.-HISI Contract of July 1, 1975.

Subject: Annual Progress Report

Enclosed is our annual progress report. The work reported therein was sponsored in part by Honeywell and the Air Force under our research and development contract for engineering of a computer system for which security can be certified by auditing.



Jerome H. Saltzer  
Head, Computer Systems Research Division

xc: N. Adleman, HISI/FSO  
L. Verdery, HISI/FSO  
R. Schell, AF/ESD  
S. Walker, ARPA/IPTO

ANNUAL REPORT TO HONEYWELL: July, 1975--June, 1976

by J. H. Saltzer, M. D. Schroeder, D. P. Reed, and D. D. Clark

The Computer Systems Research Division of the M.I.T. Laboratory for Computer Science completed several key parts of its information sharing kernel design project during the 1975-76 year. The following section from the Division's Annual Report describes the work in this area.

#### THE INFORMATION SHARING KERNEL DESIGN PROJECT

About three years ago, we entered into a subcontract with Honeywell Information Systems Inc. to perform engineering studies on strategies for simplifying the design of the resource- and information-sharing kernel of a full-scale computer system, with the goal of making the security aspects of a system simple enough that certification of correctness might be possible. Multics is the laboratory in which these experiments have been performed. This year, significant progress occurred on several key aspects of this work:

- . Development of the use of type-extension as a strategy for systematic design of the kernel itself.
- . Organization of processor multiplexing in two layers, with memory multiplexing sandwiched between, to untangle these two complex mechanisms.
- . Organization of memory multiplexing in identified parallel processes rather than in a central structure.
- . Organization of process initiation as an unprivileged operation controlled by domain entry mechanisms.
- . Development of a new model of process synchronization, called the "eventcount" model, that leads to simpler coordination algorithms and minimizes unnecessary communication, a feature important to security.

The cumulative impact of these projects on the structure of a system kernel, together with a variety of other ideas currently being explored, appears to be significant in that the kernel becomes modular, ordered, and thereby incrementally verifiable.

The activities reported this year on this project are of a different nature than those reported in previous years. Earlier reports concentrated on reducing the size of the security kernel by removing unnecessary functions, while this year's work has concentrated on better understanding of how the remaining, essential functions might be more systematically organized. Two key ideas have led us to this understanding. First, the use of abstract types as a methodology for choosing and specifying the interfaces inside the kernel (as pioneered in HYDRA, CLU, and SIMULA) gives a useful and clear decomposition of the kernel. Second, the use of processes within the kernel to multiplex the resources used in implementing objects of abstract type gives a much simpler control structure inside the kernel.

Our basic approach to simplifying the structure of the kernel is to decompose its design and implementation into modules. By structuring the decomposition into modules correctly, we hope to obtain a system in which understanding or verifying the system as a whole requires little more effort than understanding or verifying every module separately. The problem with obtaining such a well-structured decomposition of the system is to find a way to decompose the system into modules that are internally simple and have simple interactions with the other modules of the system.

Simplifying the interactions among modules is aided by two techniques. First, the method by which interacting modules communicate can be simplified. Philippe Janson, in his Ph.D. thesis, has categorized modularizations into two classes: strict modularization, in which modules interact with another module only by invoking procedures in the other module, and weak modularization, in which modules may communicate via shared data bases. By designing a system in terms of strict modules, it is much simpler to define the effect of a particular intermodule interaction. The second technique for

simplifying interaction is to define a partial ordering of modules based on functional dependency. Module A depends on module B if B must correctly meet its functional specification in order for A to meet its functional specification. If all dependencies are uni-directional, and form a partial ordering, then it can be quite simple to verify the correct operation of all modules. One starts with modules that are assumed to be correct (for example, the hardware) and proceeds to verify all modules by induction on the partially-ordered structure.

#### Abstract Types as a Structuring Tool

A structuring methodology that leads to both a strict modularization and a modularization that is partially ordered in functional dependency is the type-extension mechanism for creating abstract types. An abstract type is a collection of abstract objects and operations on the abstract objects. The specification of the properties of and interface to the objects of the type is independent of the actual storage representation of the objects or implementation of the operations in terms of the storage representation. The only way to manipulate objects of the type is to call on the operations of the type. Thus a modularization based on abstract types is strict. Types may be implemented in terms of objects of other types. This results in a uni-directional functional dependency.

Both Janson and David Reed have investigated the use of abstract types in the design of the kernel of an operating system such as Multics. In an operating system, the implementation of abstract types and the process of type extension cause difficulties not present in abstract type concepts as implemented in programming languages such as CLU. The major difficulty arises from scarcity of memory and processing resources to implement objects and operations, requiring multiplexing of those resources. Using the abstract type concept to structure the multiplexing functions has led to some new insights into the structure of operating systems and the mechanism of type extension. In contrast, the HYDRA system, which supports abstract types outside the kernel, does not use abstract types in the multiplexing of memory and processors to provide virtual memory or virtual processors.

Janson has defined a new model of abstract types to be used in the design of the kernel of the system where multiplexing of objects is the key problem. The primary difference between this model and older ones is that he explicitly recognizes the limitations on the supply of low-level resources, such as primary memory and processor resources. He also recognizes the multiplexing function, by explicitly including in his model a time-varying mapping between objects of abstract type and the objects used in their representation.

An important part of describing a modularization is to determine all functional dependencies between modules. Janson has extensively categorized these dependencies for a modularization based on abstract types. The five categories he has described are:

1. Component dependencies - dependency of an abstract type on the types used to provide storage for parts of the objects.
2. Program dependencies - dependency of an abstract type on the types used to provide storage for programs that implement its operations.
3. Map dependencies - dependency of a type on the types used to provide storage for its maps.
4. Environment dependencies - dependency of a type on the types that are used to structure the address space or naming environment of the programs that implement the type.
5. Interpreter dependencies - dependency of a type on the types used to control the allocation of processor resources to the programs that implement the type.

The environment and interpreter dependencies are particularly difficult to deal with in structuring a system. A mechanism for simplifying these dependencies has been proposed by Reed. It consists of implementing the kernel type managers on dedicated virtual processors that rely on a simple

(perhaps hardware-implemented) fixed addressing environment, rather than as operations in a privileged domain of each user process. The environment of the type manager need not depend on the domain mechanism, and the processing of type manager operations does not depend on the resource control mechanisms that regulate the virtual processors that run user processes.

A particular pattern of type-extension that recurs frequently in construction of a kernel has been described by Janson. It is the cache management pattern, which consists of building a new type of object out of two representation types, the cache type and the encached type. The functionality of the new type is quite similar to the functionality of the cache type. The encached type merely provides a large amount of storage. This pattern arises because there are not enough objects of cache type. A new type is created using the encached type to store the status of objects of the new type whenever they are not stored in cache type objects. Janson finds numerous examples of this pattern in the virtual memory design; for example, a virtual memory page type is created out of a primary memory page type and a secondary memory page type. Reed has also found this pattern in structuring the implementation of virtual processors.

#### Disentangling Processor and Memory Multiplexing

An important result of our work on structuring the kernel is actually disentangling the interdependency between processor and memory multiplexing algorithms. This interdependency results from the need to provide a large amount of memory for tables used in implementing virtual processors for user computations performed by the operating system and the simultaneous need to provide and control the processing power used to interpret the virtual memory algorithms.

The technique used by Reed to break up this interdependency is to divide processor multiplexing into two levels. The first level of processor multiplexing provides a small set of virtual processors, called level 1 processors, that have sufficient functionality to implement the virtual memory algorithms. These virtual processors access primary memory in exactly the same way that physical processors do, through address translation hardware.

Any attempt to access an object not in primary memory is reflected as a fault, just as in the real processor. The virtual memory software is implemented in terms of these level 1 processors. Andrew Huber has proposed a design for virtual memory implementation that uses multiple dedicated virtual processors to perform its functions. The second level of processor multiplexing multiplexes a subset of the level 1 virtual processors to provide a large set of level 2 virtual processors, used to run user processes. The data bases of the level 2 processor multiplexing algorithms are implemented in terms of virtual memory objects. The processor resources for the level 2 manager algorithms are provided by three dedicated level 1 processors.

#### Using Processes as a Structuring Tool

As a result of this two level design, level 1 virtual processors can be dedicated to handle management of many multiplexed operating system resources. Level 1 processors are relatively cheap compared to real physical processors, so dedicating them gives some of the effect of dedicating a physical processor, without the cost.

Structuring the kernel as a set of processes running on dedicated level 1 processors is another powerful tool for structuring the kernel. The opposite approach, used in operating systems like Multics, TENEX, and OS/360, is to implement kernel operations as subroutines called by users of those operations. Let us call the first approach the multi-process supervisor approach, and the second the distributed supervisor approach.

The multi-process supervisor approach simplifies the handling of types built of multiplexed resources by centralizing the operations that manage those resources in one or more dedicated processes. In such a design, a type manager process is isolated from the processes that request operations on the resources. Consequently, interference with the implementation of the type by processes using the type is precluded.

One advantage of implementing a type manager as a process is that it need not share a data base with other instances of itself acting in parallel. Only the type manager process need have access to the data structures used in managing the objects it implements. The sequentiality imposed by interlocking

in the distributed supervisor is achieved by using the sequentiality inherent in the queue of the type manager process. The sequence of actions that may be performed on objects is explicitly represented in the programs of the type manager process, rather than implicitly in the locking protocols.

Another advantage of implementing a type manager as a process on a dedicated processor is isolation of its environment and control point from accidental (or intentional) interference. As noted above, the environment of a type manager executing on its own dedicated processor need not be managed by the same manager that performs the complex operations needed to manage user process environments. This simplifies the dependency structure by eliminating environment dependencies. Similarly, the multiplexing of processor resources that provides resources to type managers need not include the complexity of the resource controls used to limit user process resource usage. On the other side, the implementation of user process environments and scheduling algorithms for user processes need not take into account the special requirements of user processes when executing kernel algorithms (such as protecting the process from destruction while in the kernel or protecting the kernel type manager environment from tampering). Taking these requirements into account would in any case probably result in a cyclic dependency.

The allocation of kernel type managers to dedicated level 1 processors also aids the principle of least privilege. Each type manager need have only the privileges necessary to access its own data bases. This principle can be enforced by restricting the environment (by controlling the set of descriptors in the descriptor segment) of the type manager processes. In a distributed supervisor, on the other hand, the kernel operations have access to more objects than they need. For example, in the present Multics, every kernel operation has access to all objects in the environment of the user process that invokes it. An operation that maps a page into primary memory has the capability to simultaneously copy data from one user object to another. In a distributed supervisor, for this reason, each supervisor operation must be inspected to see that it does not do additional operations extraneous to its function. The multi-process structure provides a natural mechanism for mutual protection.



Finally, the multi-process structure helps simplify the structure of the system by avoiding the need to specify unnecessary ordering constraints. An example of this can be found in the design of a multi-process page control by Huber. The page removal algorithm is only indirectly coupled to the algorithm that handles page faults. Each page fault requires using up a page frame in primary memory, but waiting until a page fault occurs to write pages out of primary memory would result in unnecessary delay. To avoid this delay, the pages that are to be written should be located and the write started by a predictive algorithm, which is very hard to fit into a page manager that is invoked only on each fault. A much better structure would be to implement the page removal algorithm as a process that controls the rate of removal of pages in a way that is only loosely coupled to the fault sequence. The page removal algorithm can then easily be designed to run at the optimal times, rather than being constrained to execute only at page fault time. This use of processes also exemplifies the principle of least privilege, because the faulting process need never touch a page other than the one it requires (and presumably has access to). In a distributed supervisor, where removal is done at fault time, the fault handler doing the removal must touch pages that the user process should not have access to.

#### Impact on the Kernel Design Project

The work of Janson, Huber, and Reed has led to a fairly cohesive and implementable kernel design. Janson and Reed have worked out a structuring of the Multics kernel into modules that each manage one abstract type. The use of processes to structure the kernel has been investigated by Huber and Reed.

The status of the use of these ideas in the design of a Multics kernel varies. Huber implemented and tested his use of processes in page control in a special version of Multics. Reed has proposed a detailed design for the two levels of processor multiplexing. A test implementation of part of this design is in progress. Janson has proposed a very detailed structure for the virtual memory management portion of the Multics kernel.

### Related Activities

In addition to the closely interrelated activities just mentioned, several other activities in the kernel design project either were completed or made significant progress during the year:

- 1) An internal report was completed by Rajendra Kanodia and Reed describing the use and implementation of the "eventcount" process coordination model. Basically, eventcounts are semaphore-like coordination variables that are constrained to take on monotonically increasing values. Coordination of parallel activities is achieved by having a process wait for an eventcount to attain a given value; one process signals another by incrementing the value of an eventcount. Any coordination problem for which a solution has been developed using semaphores can be easily converted to a solution using eventcounts. In addition many eventcount solutions seem to have the property that most eventcounts are written into by only one process; this reduction in write contention has beneficial effects on security problems and on coordination of processes separated by a transmission delay, as in a "distributed" computer system. Eventcounts provide a solution to the "confined readers" problem, a version of the reader's-writer's coordination problem in which readers of the information are supposed to be confined in such a way that they cannot communicate information to the writers. Finally, for the class of synchronization problems encountered inside an operating system kernel, eventcounts appear to lead to simple, easy-to-verify solutions.
- 2) A thesis and trial implementation completed by Warren Montgomery establish that it is practical to remove many of the traditional constraints on process creation without creating problems for security or resource administration. The concern here is that when a process is created, say in response to a user's dial-up and request for service, the designation of the principal identifier for the new process must be done correctly, or else all access control will be worthless. For this reason, process-creating programs of the "network logger", the "answering service" and the "absentee user manager" have been considered sensitive, privileged programs. Montgomery's approach is to allow any process to request creation of other processes

without restraint on principal identifiers proposed; control is provided by associating with every principal identifier a designated starting procedure for the new process. This starting procedure checks to see if proper identification has been submitted by the requestor of the creation. By decentralizing this check, making it the responsibility of the concerned party, a strategy parallel to that of entering a protected subsystem (at a designated starting point) has been created. The result is to remove from the security kernel several large programs previously thought to require certification.

3) The use of end-to-end cryptographic protection for network connection to a secure host was explored in depth by Stephen Kent in an independently supported, but closely related, project. Kent examined the impact of end-to-end encryption on network protocols, and developed strategies for character-at-a-time full duplex interaction, key distribution, and resynchronization following high-priority messages or line disruption. He also examined the question of proper placement, within an operating system, of a cryptographic protection module. He concluded by developing a practical design, based on the National Bureau of Standards Data Encryption Standard, and testing that design in a Multics/ARPANET implementation.

4) Another related activity, supported by Honeywell and Ford Motor Company, was the trial, by David Gifford, of a simple method of estimating the primary memory requirement of an executing program, for control of multiprogramming. Gifford's method is to observe the rate of "misses" of the processor's associative memory for page table words, and assume that a high miss rate is an indication that a large program is being executed. Gifford found that basing multiprogramming control on this measurement provided a level of system effectiveness equal to that achieved by careful hand tuning, and that the incredibly complex memory size estimator currently in the Multics security kernel is unnecessary.

With the completion of the activities described above, the majority of work planned for the kernel design project is finished. We expect that the coming year will see the completion of the remaining research tasks for this

project, and a final report; activity will continue, however, to provide support and technology transfer to the larger Air Force/Honeywell project of which this work has been a part.

## PUBLICATIONS, TALKS, AND THESES

### Publications

- Redell, D.D. and Fabry, R.S., "Selective Revocation of Capabilities," submitted to Communications of the ACM, April, 1976.
- Redell, D.D. and Clark, D.D., "Protection of Information in Computer Systems," Tutorial notes, IEEE Publication No. 75CH1050-4, September, 1975.
- Saltzer, J.H. and Schroeder, M.D., "The Protection of Information in Computer Systems," Proceedings of IEEE 63, 9, (Sept., 1975), pp. 1278-1308.
- Saltzer, J.H., "Computer," article in 1976 Yearbook, McGraw-Hill Encyclopedia of Science and Technology New York: McGraw-Hill (Scheduled 1976).
- Schroeder, M.D., "Engineering a Security Kernel for Multics," Proceedings of 5th Symposium on Operating Systems Principles, ACM Operating Systems Review 9, 5, pp. 25-32.

### Theses Completed

- Gifford, D., "Hardware Estimation of a Process' Primary Memory Requirements," S.B. Thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, May, 1976.
- Bratt, R., "Minimizing the Naming Facilities Requiring Protection in a Computer Utility," S.M. Thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, July, 1975, also Project MAC Technical Report TR-156.
- Huber, A., "A Multi-process Design of a Paging System," S.M. and E.E. Thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, May, 1976.
- Kent, S., "Encryption-Based Protocols for Interactive User-Computer Communication," S.M. Thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, May, 1976, also Laboratory for Computer Science Technical Report TR-162.
- Montgomery, W., "A Secure and Flexible Model of Process Initiation for a Computer Utility," S.M. and E.E. thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, June, 1976.

Reed, D., "Process Multiplexing in a Layered Operating System," S.M. thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, June, 1976.

### Theses in Progress

Goldberg, H., "Protecting User Environments," S.M. thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, expected date of completion, November, 1976.

Hunt, D., "A Case Study of Intermodule Dependencies in a Virtual Memory Subsystem," E.E. thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, expected date of completion, September, 1976.

Luniewski, A., "A Certifiable System Initialization Mechanism," S.M. thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, expected date of completion, January, 1977.

Janson, P., "Using Type Extension to Organize Virtual Memory Mechanisms," Ph.D. thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, expected date of completion, August, 1976.

Feiertag, R., "A Methodology for Designing Certifiably Secure Computer Systems," Ph.D. thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science.

### Talks and Presentations

Kanodia, R., "Eventcounts: A new model of process synchronization," given at: Institute for Advanced Computation, Sunnyvale, California, August, 1975.  
Xerox Palo Alto Research Center, June 14, 1976  
IBM, Thomas J. Watson Research Center, Yorktown Heights, New York, June 24, 1976.

Redell, D.D. and Clark, D.D., "Protection of Information in Computer Systems," day-long tutorial given at:  
Eleventh IEEE Computer Society Conference, Washington, D.C., September 8, 1975.

Hunt, D., "A Case Study of Intermodule Dependencies in a Virtual Memory Subsystem," given at:  
Sperry Research Center, Sudbury, Massachusetts, October, 1975.  
C.S. Draper Laboratory, Cambridge, Massachusetts, April, 1976.

Redell, D.D., "Proprietary Subsystems and Personal Computers," given at:  
IBM San Jose Research Laboratory, November 17, 1975.  
Xerox Palo Alto Research Center, February 6, 1976.

Clark, D.D., "Engineering a Security Kernel for Multics," given at:  
University of Southwestern Louisiana, November 18, 1975.

Redell, D.D., "The Multics Kernel Design Project," given at:  
IBM San Jose Research Laboratory, March 5, 1976.

Schroeder, M.D., "The Multics Kernel Project," given at:  
Xerox Palo Alto Research Center, January, 1976.  
Cambridge University, England, April, 1976.

Janson, P., "Validating the Protection Mechanism of a System," given at:  
IRIA Workshop on Protection and Security in Data Networks, France, June  
28, 1976.

Committee Memberships

Saltzer, J.H., ARPA Secure Systems Working Group

PERSONNEL, June, 1975 - July, 1976

Faculty and Research Associates

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

Professional Staff

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

Support Staff

Pauly G. Heinmiller  
 Virginia M. Newcomb  
 Carol Sarnier  
 Muriel Webber

Undergraduate Students

David K. Gifford  
 Arthur G. Gottlieb  
 Barry M. Grant

Graduate Students

Toby Bloom  
 Eugene C. Ciccarelli  
 Richard J. Feiertag  
 Harry C. Forsdick  
 Robert M. Frankston  
 Harold J. Goldberg  
 Andrew R. Huber  
 Douglas H. Hunt  
 Philippe A. Janson  
 Stephen T. Kent  
 Allen W. Luniewski  
 Andrew A. Montgomery  
 David P. Reed  
 Victor Voydock

Guests

Nathan A. Adleman  
 William B. Maczko