COMPUTER SYSTEMS RESEARCH DIVISION ANNUAL PROGRESS REPORT July, 1973 to June, 1974

by Jerome H. Saltzer, Michael D. Schroeder, and David D. Clark

OVERVIEW

The current research activities of the Computer Systems Research Division can roughly be described as trying to discover and implement the minimum mechanism essential to support a full-scale computer utility system. Its activities are pragmatic, which means that most ideas are subjected to practical implementations as part of their development. The division uses the Multics system as its laboratory, taking advantage of the nearby Honeywell Multics development facilities to test special modified versions of the system.

The largest portion of current work is inspired by the need to certify the correctness of privacy-achieving and other information-isolating mechanisms in a shared-user system. On the basis that certification of correctness should be easier if the mechanism being certified is simpler, work is proceeding to first identify and then minimize the complexity of the central protection kernel of Multics.

A second major area of interest is simplifying and better understanding the attachment of the ARPANET to Multics. Because the ARPANET involves an element of distributed computations connected by communication lines, it relates directly to a longer-range interest in the future of computer-utility systems in an era when logic and memory costs are predicted to make personal computers as commonplace as today's hand-held calculators. The central utility is still needed for communication, sharing of information, and handling peak loads, but its interfaces and possibly its functions must certainly be different from those of today's systems.

Another activity reported here is called "technology transfer", a collection of efforts to communicate with industry and users the results of previous research, particularly the development of the Multics system.

This report is organized in four sections. The first three describe the major work: the certification/simplification projects, ARPANET-related activities, and the technology transfer activities. The fourth reports other miscellaneous activities of the division.

CERTIFICATION OF COMPUTER SYSTEMS

Introduction

This year the Computer Systems Research Division began a new research project with the goal of making possible the certification that the data security facilities in a large-scale, multiuser computer system have been correctly implemented. This effort, intended to be the primary research activity of the Computer Systems Research Division for approximately three years, is directed at the problem of producing computer systems which guarantee to prevent unauthorized release, modification, and denial of use of the information that they contain. The need for such certification arises when a single system provides computation and information storage service to a community of users. As the economic and functional advantages of such shared systems have been recognized, so has the need to include facilities for controlling the access of the various users to the contained information. Without these facilities, sensitive information can be handled only if the user community is carefully restricted to be a highly homogeneous group. Many systems now include protection mechanisms for enforcing intricate, externally specified policies on information access. The presence of such mechanisms, however, is not enough. Users, whether they be individuals or private or governmental organizations, must have confidence in the integrity of the protection mechanisms before they can entrust sensitive data to a system. The system must be certified to implement without failure the desired policies for controlling access to the contained information.

There are three ways in which the security of information stored in a computer system can be violated:

1. Unauthorized release: an unauthorized person is able to read, and take advantage of, information stored in the computer. Concern sometimes extends to "traffic analysis", in which the person observes only the patterns of use of information and from those patterns can infer some content.
2. Unauthorized modification: an unauthorized person is able to cause unexpected changes to stored information.
3. Unauthorized denial of use: an unauthorized person can prevent legitimate access or modification, even though he may not be able to access or modify the information, for example by causing a system "crash".

Complicating things in a shared computer is the fact that the unauthorized person with respect to a specific act may be an otherwise legimate user of the system.

In practice, producing a system that actually does prevent all such unauthorized activities has proved extremely difficult. Sophisticated users of most currently available systems are probably aware of at least one way to "crash" the system. Penetration exercises involving a large number of different systems have shown that, in all systems confronted, a wily user can construct a program that can obtain unauthorized access to information stored within the system.

The primary reason for these failures is the presence of design and implementation flaws that provide paths by which the access constraints supposedly enforced by the system can be circumvented. Underlying this cause are two interacting difficulties. The first is that preventing all unauthorized acts is a negative kind of requirement. It is intrinsically quite hard to prove that this requirement has actually been achieved, for one must demonstrate that no means for violating data security exist. The second is

the well-known tendency for the operating systems of shared, general-purpose computers to be extraordinarily complex, large in size, difficult to maintain, and awkwardly organized. This tendency interacts badly with the need to prove non-existance of paths for violating data security, by providing a very complex environment in which to attempt such a proof.

There seem to be several reasons for the tendency toward complexity, such as:

- attempts to stretch the functional capabilities of the system as far as possible;
- working in a hardware environment that was determined before software requirements were fully understood;
- attempts to squeeze the system to its absolute limit of performance;
- attempts, because of the high cost of system development, to get the system running in the absolutely shortest time possible.

Of these four, probably the last two are the strongest contributors to overall complexity, since both encourage shortcuts to be taken and modularity to be violated against the better judgement of the system designer.

The certification of a system means that someone has signed-off on a statement of adequacy. By signing, the certifier states that the security provided is adequate to the intended application. He also assumes the responsibility for failures. A system is certifiable if the certifier can be convinced to sign. With currently available commercial systems, there is no way for a potential certifier even to start developing the confidence in a system prerequisite to signing. Most are of a size and complexity to preclude even reading all of the code, much less comprehending the entire mass in detail.

The Computer Systems Research Division research effort is aimed directly at the size and complexity. The overall plan is to evolve an existing, commercial, multiuser computer system, Multics, which is easily modifiable and which has advanced protection mechanisms, into a prototype operating system with all the essential features of the present Multics system, but with a small and simple central core that is susceptible to certification through line-by-line review by an expert. The goal is a system sufficiently small, well-structured and easy to understand that a certifier can read it all, understand the reason for every line of code, and develop a confidence in its correct operation adequate to most applications.

Method of Attack

The problem of constructing a certifiably secure system recently has attracted considerable interest and is being attacked with a variety of different strategies by many research groups in addition to the Computer Systems Research Division [Saltzer: "Ongoing Research and Development on Information Protection", ACM Operating Systems Review, July, 1974]. It is generally recognozed that the key to the ultimate solution is methodical design and construction techniques which systematically exclude flaws that can be exploited to produce security violations. Many imagine ultimately being able to construct a formal specification for a system, prove desired security (and other) properties about the specification, and then, by essentially mechanical steps, construct a matching operational system. To this end, many research groups are conducting investigations into methods of proving assertions about programs and program-like specifications, methods for formally describing properties like security, and techniques of top-down program construction by successive refinement of descriptions of algorithms and data structures. On the other end of the

spectrum, several groups are engaged in finding and cataloguing flaws in existing systems with the aim of convincing skeptics that the problem is real and of understanding the sort of flaws that can be exploited. Somewhere between these two extremes are several groups, including our own, looking for the simplest possible structures with which to securely implement the full set of functions that seem desirable in a multiuser, general-purpose computer system. An understanding of simpler ways to organize such systems will contribute to the development of the ultimately required mechanical construction techniques. But of more immediate importance, it will also allow us to build less complex systems to do the same job, thus providing a less complex environment in which to establish the absence of security flaws.

Given that problems of structure are to be attacked, two approaches are possible. This first is to wipe the slate clean and design a new system from scratch. The accumulated knowledge of past successes and failures could be brought to bear in an attempt to produce a new design that is well-organized, simple, and concise. By starting from scratch a great deal of freedom is gained to organize the entire system and its specifications to facilitate demonstration of a lack of flaws. The second approach is to modify in an evolutionary way an existing system so as to simplify its organization to the point where the absence of security flaws can be demonstrated. Our choice of this second approach requires some comment.

The first approach, while appealing, has the defect that there seems to be no way to release the designer of a new system from the pressures toward complexity which were mentioned earlier, especially the pressure to get a system operational as soon as possible because of the development expense. Any attempt to mitigate this pressure by stopping short of producing

an operational system seems to have two problems. First, with the current state of understanding of computer systems, it is hard to have confidence that the full implications of a system structure are understood without complete implementation. Second, if an operational system is not the goal, it is very easy to leave out many of the complexity-producing convenience features that users demand of a production system. A structure which gracefully supports a toy system may be badly strained under the load of conflicting features required in its real decendant. Put another way, design and implementation flaws representing potential security violations tend not to be a problem in toy systems.

To avoid these difficulties, the Computer Systems Research Division has adopted the approach of evolving an existing operating system to simplify its structure and reduce its bulk. The great danger of this approach is that the system chosen for evolution will prove so resistant to graceful alteration that no evolution of structure is possible, short of starting over. Thus, it is extremely important to pick a suitable subject. We are using the Multics system, previously developed by the Computer Systems Research Division of Project MAC. Multics is better organized than most systems for evolution and modification, because it is relatively modular, is largely written in PL/I, and was originally constructed with evolution as a primary objective. Also, Multics has been developed from the ground up to protect the information it contains from unauthorized access. It already includes protection mechanisms as advanced as any available, including special hardware features such as protection rings. Thus, the system both exhibits a set of protection features that would be interesting to certify and provides protection features that will make the job of certification easier. Finally, because Multics is a

commercially available product and new ideas developed in the course of this research should be relatively easy to retrofit to the standard system, the result, if successful, can be easily exported in a directly useful way.

Although the original design of Multics was very methodical, and the system is, if anything, already less complex in organization than most contemporary computer operating systems with similar functional goals, potentially, it could be supported with mechanisms that are much simpler yet. The intense pressure of initial implementation did not permit time for contemplation and development of simpler supporting structures. The basic premise of this research is that one wave of simplification applied to the central core of the system will produce a badly needed example of a structure that is significantly easier to understand.

The Security Kernel

The total volume of software in a system like Multics is enormous. In addition to the supervisor and other system provided software such as subroutine libraries, compilers, and specialized applications packages, a large community of active users produces many programs of its own. If the security of information in such a system depended upon the correctness of the entire collection, then our task clearly would be hopeless. To make progress, the system must be arranged so that the security of each user's data depends only on the correct operation of some subset of all the software contained in the system--the smaller this subset the better. Indeed, the programs produced and executed by other users must be able to be excluded from the subset affecting any one user, for the potential malicious activities of another user is the presumed threat.

The overall structure used by Multics to control user access to stored information is to provide each user computation with its own process and address space. A process has no ability to access the address space of another. In the address space of every process are the procedures and data of the Multics supervisor. Among other things, the supervisor manages hardware resources and creates processes and their address spaces. The ring protection mechanism of the hardware processors is exploited to restrict the access to the supervisor of the user code executing in a process.* Supervisor components cannot be directly referenced; only specially designated entry points may be called. Once called, the supervisor procedures have direct access to other supervisor procedures and data. The supervisor manages all on-line storage, and can be requested to add a segment of on-line data

* The rings actually provide a process with eight different protection states rather than the two implied here, but that level of detail is unnecessary for this discussion.

or procedure to a process's address space where it may be referenced by the user code of that process. Such a request will be granted only if the supervisor has been informed that the user controlling the process is authorized to access the segment.

The security of the data stored in the system certainly depends upon the correctness of the supervisor, for it is the primary path by which one user's computation can influence another's data or computation, legitimately or otherwise. For example, an attempt by one user to gain unauthorized access to data, or to deny access to an authorized user, might be made by invoking a supervisor entry with an unexpected pattern of arguments, perhaps causing the supervisor mistakenly to do a dirty deed to another user. Because of the size and complexity of the supervisor, the chances of a clever attacker ultimately succeeding in uncovering an exploitable flaw are good.

The supervisor is a software mechanism that is common to all users of Multics. A mechanism is common to a group of users if it can be used by one to influence the data or computation of another in the group legitimately or otherwise. At the heart of every common mechanism must be some group of data items whose value one user's computation can influence and another's can notice. The influence and notice may be very direct--one writes into a data item and another reads it--or quite indirect--the invocation of a procedure by one somehow alters its internal state so that the outcome of a later invocation by another is affected. Common mechanisms are required to implement any explicit or implicit communication among a set of users. If no such communication or coordination is involved, however, then a common mechanism is not required to implement a function. It is precisely the existence of common mechanisms that allow one user the possibility of exerting unauthorized influence over the computations or data of another. Malicious users must

exploit flaws in common mechanisms to work their will. To prevent such malicious activity it is the common mechanisms that must be certified to contain no exploitable flaws, and once certified must be protected against tampering.

The Multics supervisor is bigger and more complex than it needs to be. This is partly the result of the presence in the supervisor of procedures and data providing functions that need not be implemented with common mechanisms, for they include no element of communication or coordination. Yet, by being part of the supervisor, with the attendant access privileges, effectively they are part of the common mechanism. Flaws they contain may be exploited as illicit access paths.

A primary strategy of the research project is to evolve the current supervisor into a security kernel that contains only functions required to be implemented as common mechanisms, removing all other functions to execute as user programs in each process, where they cannot be exploited as illicit interuser access paths. The security kernel produced should be the least amount of common mechanism necessary to implement the patterns of information sharing, interprocess communication, and physical resource multiplexing that are required in the system. As the common mechanism is made small its structure will be simplified also, the goal being a smallness and simplicity sufficient to permit certification of the resulting kernel. It appears feasible to extract a kernel with 4,000 to 8,000 lines of source code from the present supervisor of approximately 50,000 lines of source code.*

* It is expected that almost all of the source code will be in the PL/I language. Using PL/I to generate the kernel seems to require that the PL/I compiler be certified, as well as the kernel, a troubling thought since the compiler itself is a 25,000 line PL/I program. In the case of the compiler, however, certification may be less of a problem than for the kernel. The kernel needs to work correctly for all possible input; the compiler need compile correctly only the specific programs of the kernel -- not all possible programs. Thus, the compiler's effect on the kernel can be certified by comparing the source code for each kernel module with the compiler-produced object code, a task much simpler than certifying the compiler correct for all possible source programs.

Does a certified security kernel, as just defined, really produce a system guaranteed to prevent all unauthorized attempts to release, modify or deny access to contained data? The answer, unfortunately, is no. Any non-security kernel software which executes in a process that has access to some data has the potential to compromise that data. These programs can be grouped in four categories. First there are the system-provided programs--the library subroutines, compilers, and application packages available in most systems plus all the programs thrown out of the old supervisor in the minimization of the new security kernel. These system-provided programs are not common mechanisms, even though in Multics all processes share the same non-writeable segments of code that embody their algorithm, for a private copy of the alterable part of these procedures, the variable data, is provided for each process. Because they are private mechanisms, no interuser interaction can occur through them. They may still contain errors if they are not certified, but these errors can be triggered only by the actions of the process that they might damage as a result of the triggering. By presuming that the system programmers who constructed them are non-malicious and did not willfully plant "trojan horses", it seems justified to assume that the mistakes caused by these system-provided procedures will decrease in time as all normally used functions are exercised, and that the security threat posed by a potential random error causing undesired release or modification or a users' data is acceptable for most applications. Unlike the software mechanisms of the security kernel, these are not susceptible to willful exploitation by other users. In any case, a user unsatisfied with their trustworthiness may, in Multics, choose not to use them, substituting his own procedures.

The last comment suggests the second category of procedures executing in the user environment of a process -- procedures constructed by that user. Any

security threat posed by errors in these is the user's own problem. The only possible help would be providing tools to aid the user in certifying his own programs.

The third category, possible in Multics, is procedures borrowed from other users. These are a real danger to the security of the borrower's data. Because they will execute with all the access authority of the borrower's own procedures, they can contain "trojan horse" code maliciously constructed to cause a security violation.* A user should only borrow procedures from another when the borrower has reason to trust the lender. The inclusion of security kernel facilities to support user-constructed protected subsystems provides a tool to reduce the potential damage such a borrowed trojan horse can do, but a user initiated certification of the borrowed program is the only complete protection against this threat.

The fourth category is common mechanisms set up among a group of users by their mutual consent to implement some function involving interuser communication or coordination. Such a mechanism makes the group susceptible to undesired interaction in the same way that an uncertified supervisor does for the whole user community. If a user agrees to become party to such a common mechanism, then he must satisfy himself of its trustworthiness.

In considering these four categories, it is apparent that none is so important to system security as the common mechanism of the security kernel, for every user of the system is forced to rely upon it. Because it appears to have maximum leverage on the security problem, the Computer Systems Research Division is concentrating on abstracting the security kernel and on simplifying its internal structure. A Multics with a certified security kernel would provide a usefully greater level of security than the present system provides.

---

* Note that this is a special case of a common mechanism. The data items whose value the lender can cause to change and thereby influence the data of the borrower is the code of the lent procedure itself. Even if the procedure is non-writable when lent, it was written by the lender when constructed.

Specifics

The effort by the Computer Systems Research Division to produce a security kernel for Multics can be broken into four interrelated categories of activity: reviewing, removing, simplifying and partitioning. This section describes these categories, giving examples of each. The next section provides a complete list of the work performed during this report period.

The review category covers all efforts to understand better the specific problems of the current Multics supervisor. In addition to trying to understand the reasons for the size and complexity of the current supervisor, an effort is being made to identify and correct existing security flaws. A list of all known Multics security flaws is maintained. Each flaw reported is analyzed to determine how it happened, how it can be fixed, and how similar flaws can be avoided in the future. Several audits have been made to uncover suspected new flaws. One highly successful search for new flaws was undertaken as a result of a suggestion by Richard Bisbey II, at the Information Sciences Institute of the University of Southern California, who has been trying to abstract from many system penetration exercises some general patterns that lead to security flaws in different systems. He reported that multiple references by supervisor code to user-provided arguments can be exploited in many systems to cause supervisor malfunction, and pointed out one example he had uncovered in Multics. The problem is illustrated by the following procedure which might be used to implement the segment deletion function in a system like Multics:

```
delete_seg:  procedure(name, code);                          1
             call verify_permission(name, "delete", code);   2
             if code = proper_access                         3
                then call delete(name, code);                4
             return;                                         5
end;                                                         6
```

Assume that "name" and "code" are user-provided arguments passed by address. The first identifies the segment to be deleted from the file system while the second provides a place for a return error code. The call to "verify_permission" verifies that the user controlling this process has permission to destroy the named segment. If the "code" returned indicates that the user has delete permission for the segment, then the "delete" procedure is called to perform the requested act. Now imagine that the user contrives to change the value of "code" between lines 2 and 3. Then clearly he can cause the deletion of a segment for which he has no delete permission. The same problem exists with the "name" argument. In almost all systems, including Multics, the user can cause such a carefully timed change in the value of an argument in a methodical way.*

As a result of Bisbey's suggestion, an audit of the 170 entries to the Multics supervisor was made, looking for this pattern. The audit uncovered 50 entries that made multiple references to arguments. Of these 8 clearly were exploitable security flaws, 34 looked safe, and 8 were questionable. The problem can be systematically avoided by requiring all supervisor entries to copy their arguments before using them.

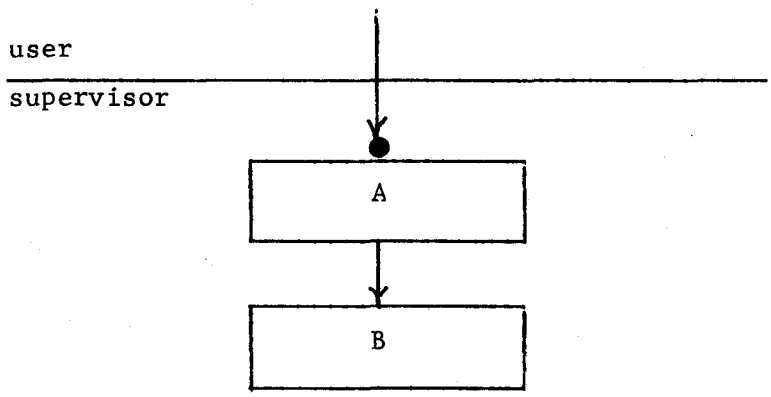
So far, all of the flaws uncovered by the review activities are isolated and easily repaired. No major design flaws have been found.

The second category of activity is removing from the supervisor those mechanisms not implementing functions of information sharing, interprocess communication, or physical resource management, i.e., those functions not

---

* In fact, it is particularly easy on the Multics Honeywell 6180 processor, because of the existence of a mode of addressing where the value of an indirect address stored in memory will increment automatically with each use. Placing an auto-incrementing indirect address in an argument list of a supervisor call can generate the desired change of argument value at just the right moment.

required to be implemented as common mechanisms. In many cases removal involves undoing a pattern caused by a performance characteristic of the Multics implementation for the Honeywell 645 computers. For that older machine protection rings were simulated in software and cross-ring calls were quite expensive. Thus, a call that went from a user protection environment to the supervisor cost much more than a call which did not change protection environments. The result was an effective pressure to include many functions in the supervisor that did not need to be implemented as part of a common mechanism. The reason for this pressure can be seen from the following figure:



A and B are procedure modules in the supervisor. Imagine that a single invocation of A (by a user procedure) can result in a flurry of calls from A to B. Then there is a clear performance cost in moving the user/supervisor boundary to between A and B, even if only B need be part of the protected, common supervisor.

The new hardware base for Multics, the Honeywell 6180, implements the protection rings in hardware. One result is that calls from one ring to another now cost no more than calls inside a ring. Thus, the performance penalty associated with supervisor calls has been removed, and many modules included in the supervisor for performance reasons rather than protection reasons now

can be removed.*

The actual removal activities are much more complex than suggested by the example of the previous paragraph. In most cases the common and private parts of a facility are not so neatly packaged in separate procedures, but are intricately intertwined in the same procedures and data bases. Insight and ingenuity are required to separate the private and common parts of a mechanism, leaving a reasonable interface. Also, supervisor procedures execute in a slightly different environment than other procedures. Code written for the supervisor environment often depends upon the special way in which the supervisor execution environment is initialized, the availability of internal interfaces implementing powerful but primitive operations, and the ability to access all segments in the address space of a process, regardless of the protection rings. Even if a module implementing only a private function were found, it might not execute outside the supervisor environment without being carefully modified.

This year the most important removal activities have been centered on the file system. In a project now almost completed the functions of dynamic intersegment linking and directing the search of the file system to satisfy a symbolic reference have been removed from the supervisor. This project is notable for two reasons. First, it removed an especially vunerable and complex mechanism from the supervisor. The vulnerability is a result of the linker having to accept user-constructed code segments as input data; the chances of such a complex "argument", if maliciously malstructured, causing the linker to malfunction while executing in the supervisor were demonstrated

---

* There may still exist other performance penalties associated with removing functions from the supervisor that will inhibit production of the smallest possible kernel. One goal of the research is to understand better the performance cost of security.

to be very high by numerous accidents. The complexity is apparent in that the linker's removal eliminated 10% of the gate entry points into the supervisor. The second interesting result of the linker's removal was the demonstration that linking procedures together across protection boundaries, i.e., rings, could be done without resort to a mechanism common to both protection regions.

A second project related to the file system is the removal from the supervisor of the facilities for managing the association between names and the segments in the address space of a process. This project, now in its initial implementation phase, requires that a data base central to the management of the address space, the known segment table, be aplit into a private and a common part, and that the supervisor learn to lie convincingly on occasion about the existence of certain file system directories. The project will result in a new, simpler interface to the file system portion of the supervisor. Instead of identifying a directory by the sequence of character string names locating it in the directory hierarchy, a segment number for the directory will be used. The notion of a tree name locating an element of the hierarchy is thus removed from the supervisor to be implemented by procedures executing in the user protection environment. (The actual file system hierarchy still remains protected inside the supervisor).

Another removal project of a different flavor is investigating the possibility of moving most of system initialization from executing inside the supervisor each time the system is started to executing once in a user environment of a previous system. The idea is to produce as a system tape a bit pattern which, when loaded into memory, manifests a fully initialized system, rather than letting the system bootstrap itself in a complex way

each time it is loaded from a tape containing the separate pieces. One pattern of operation may be much simpler to certify than the other.

The third category of activity is simplifying those mechanisms that must remain in the kernel. Such activities can reduce both the size and the complexity of the kernel. Simplification activities cover a broad range. In some cases a piece of the kernel can simply be eliminated because its function can be duplicated by another kernel mechanism. For example, the possibility of replacing all mechanisms for performing external I/O (to terminals, tape drives, card readers, card punches, and printers) with the ARPA Network attachment is being explored. This would remove from the kernel a large bulk of special mechanisms for managing the various I/O devices, leaving behind a single mechanism for managing the network attachment. Using network technology to provide the only path for external I/O to Multics appears feasible. Internal I/O functions (for managing the virtual memory, performing backup, and loading the system) would still be managed in the kernel.

Another example of simplification involves a less obvious duplication of mechanisms. A new buffering strategy for input and output from the network has been devised which, by utilizing the virtual memory, provides a core resident buffer which appears to be of infinite length. The infinite buffer scheme is much simpler than the old circular buffer which had to be used over and over again, with attendant problems of old messages not being removed before a complete circuit of the buffer was made. The old buffer scheme was really providing a special purpose storage management facility, and the simplification was to use the standard storage management facility of the system--the virtual memory--for this function.

Several specific simplification projects involve using multiple parallel processes to implement kernel functions. A characteristic of the current Multics implementation is that processes are relatively expensive, for each must have an independent address space. As a result, many system functions involving inherently parallel activities are forced into sequential algorithms. The cost is increased complexity. As a basis for reimplementing such functions taking advantage of their natural parallelism, a facility providing low cost processes is being implemented. The processes are made cheap by having several of them share the same address space. Several applications of cheap processes are underway. Each interrupt handler will be assigned its own process in which to execute, rather than being forced to inhabit whatever user process was running when the interrupt occurred. As a result, the system interrupt interceptor will simply turn each interrupt into a wakeup of the corresponding process. By virtue of being full-fledged processes, the interrupt handlers can use the normal system interprocess communication mechanisms to coordinate their activities with one another and the user process, greatly simplifying their structure.

Another important application of low cost processes is in simplifying the structure of system resource management algorithms. The mechanism for moving pages among the three levels of the memory hierachy is a good example. Whenever a missing page fault occurs in a process, the fault handler attempts to initiate the transfer of the desired page from bulk store or disk to core. This can only be done if a free core block is available. If not then the fault handler first must move a page from core to the bulk store to make room. This, in turn, is possible only if a free block of bulk store is available. If not, a page must be moved from the bulk store, via core, to a disk by the fault handler. This complex series of steps occurs sequentially with page control executing in the process which took the page fault and then in various other user processes that happen to

receive the subsequent I/O interrupts. The new scheme involving multiple dedicated processes is much simpler. One process runs in a loop making sure that some small number of free core blocks always exist. Whenever the number of free core blocks drops below that number, this process is awakened to transfer pages to bulk store. Another keeps space free on the bulk store by moving pages to disk when required. The core freeing process is activated by wakeups for processes that have taken a page fault and discovered a lack of free core blocks. The bulk store freeing process is driven in a similar manner by the core freeing process. The path taken by a user process on a page fault is greatly simplified. This process can just wait until a core block is free and then initiate the transfer of the desired page into core. The overall structure looks as though it will be much simpler than that currently employed.

The various simplification activities will eventually extend to all parts of the kernel, and to the overall structure of the kernel. Careful attention will be paid to the proper modularization of the entire kernel.

The final category of activity is partitioning the kernel into differently protected pieces that can be certified separately, some perhaps less carefully than others. While the specific projects in this category are less well developed than those for other categories, two techniques for partitioning seem worth exploring. The first is dividing the kernel that is part of each process into multiple layers in different rings of protection. For example, the bottom layer might implement a file system in which all segments were named by system generated unique identifiers. The next layer would implement a user named directory hierarchy on top of the primitive first layer file system. Another suggestion is that mechanisms to provide absolute compartmentalization of users and stored information be implemented at the bottom layer, and mechanisms to allow controlled sharing within the compartments be implemented at the next layer.

This last suggestion is particularly intriguing, because if correctly done the notion of minimizing common mechanisms would be well supported. The second layer mechanisms would be common only within each compartment.

The second partitioning technique under investigation is separating the policy component from the mechanism component of resource management algorithms by putting the policy algorithms in a non-kernel protection ring in special system processes. For example, the process described earlier that removed pages from core memory could be arranged as a multi-ring process. In the most privileged rings would execute the standard system kernel with some special gate entry points to implement movement of a particular page from core to a particular free block on the bulk store. Other gates would provide usage information on pages in core. In a less privileged ring would execute the policy algorithm that decides which page to remove when another free core block needs to be generated. The special gates into the supervisor would be used to do the actual moving, once a decision was made. The policy algorithm, however, could never read or write the contents of pages, learn the segment to which each page belonged, or cause one page to overwrite another, for the supervisor gates would be programmed to prevent these actions. The result is that the policy algorithm could never cause unauthorized use or modification of the information stored in the pages. It could only cause denial of use. Under the circumstance that denial of use was deemed less serious than the other security violations, the policy algorithm need not be as carefully certified as the rest of the kernel. It appears that the idea of separating policy from mechanisms applies to all resource management algorithms.

This completes the discussion of the specific techniques to be used by the Computer Systems Research Division to produce a certifiably secure kernel for Multics. The next section details the specific tasks performed during the progress report period.

Tasks in the Certification Project, July 1, 1973 - June 30, 1974

The following lists all tasks on which progress was made during the year. Since the list is complete, some of the tasks mentioned here duplicate the samples of the previous section.

I. Census of Ring 0.

As a first step in the certification of the Multics system, it was necessary to get some rough idea of the magnitude and structure of the present kernel of the system. To provide this information, Victor Voydock prepared a summary of the size of all the ring 0 modules, and listed these modules according to what subsystem they were a part of, and according to their source language. This overview of the present system was very helpful in determining which components of the system ought to be attacked first.

II. Removal of the Linker from Ring 0.

The project of removing the linker from ring 0 was undertaken by Phillipe Janson as a Master's thesis, which was completed in May, 1974, and is now available as a Project MAC Technical Report, TR-132. It was important that the linker be removed, since several other components of the system, in particular the management of reference names, could conceivably be removed from the kernel after the linker had been removed. The user ring version of the linker which he created is now completely operational; the only task remaining before the linker can be installed in the standard system is to insure that performance of the new linker is at least equivalent to the performance of the linker currently being used in the kernel.

III. Removal of Name-Space Management From Ring 0.

One of the functions of the Known Segment Table, or KST, is to remember the associations between segment numbers and reference names on a per-process basis. One of the principal users of this facility is the linker, which must resolve named references into segment numbers. With the linker now existing in the user ring, it is possible to consider removing portions of the KST manager into the user ring as well. Richard Bratt, as part of his Master's thesis research, has proposed a scheme for moving this function to the user ring which eliminates the drawbacks of allowing the user to directly initiate directories. The only function which remains inside the kernel of the system in his proposal is the association between segment numbers and unique ID's.

IV. Removal of the Storage Hierarchy from Ring 0.

The two previous tasks represent removal from the kernel of the system of much of the per-process segment name management. Douglas Hunt is also considering whether certain of the system wide name management mechanism could be removed from the kernel or at least partitioned in a separate area of the kernel. In particular, he is considering whether the concept of the storage hierarchy could be removed from the kernel, leaving within the kernel only a catalog of segments indexed by unique ID. In the outer ring an association would be maintained between name, unique ID, and segment number.

V. Removal of User I/O from Ring 0.

A thesis completed by David Clark, now available as Project MAC Technical Report TR-117, discusses a strategy for handling user-initiated I/O which operates almost completely in the user ring. The only function which is required within the kernel is the management of multiplexed devices. The scheme uses as the buffering strategy for I/O the virtual memory management algorithm of the system itself. The scheme described in the thesis effectively removes I/O from the kernel of the system; however, it requires an I/O controller with capabilities slightly greater than the one currently available on Multics, so that this particular removal will not actually be implemented in the near future. However, Honeywell has implemented an interface to the I/O system, which has some of the same features as the scheme described in the thesis.

VI. Simplification to Page Control.

Andrew Huber is considering ways to simplify the memory management algorithm of the Multics system. In particular, he is considering a reorganization in which most of the functions of page control are executed in separate asynchronous processes, so that the only task which the user process executes is the actual fetching of a missing page. It is felt that by isolating functions in separate processes, and constraining them by restricting the interprocess communication paths, it will be easier to understand and certify the overall algorithm. One of the other benefits of structuring page control in this way is that it should be possible for several processors to take and handle a page exception simultaneously, without interfering with each other. We are currently preparing, in a high level language devised by Bernard Greenberg, a version of page control structured in this fashion, which can be used for discussion.

VII. Simplification of Traffic Control.

The group is attempting to restructure the process scheduling algorithm of the Multics system. In particular, we are interested in separating two ideas, the actual switching of the processor from one process to another, and the decision making algorithm which determines which processes are eligible to run. It is hoped that this will speed up the act of switching from one process to another, and also make the algorithm easier to understand.

VIII. Removal of the Answering Service from the Kernel of the System.

The Answering Service, those algorithms which authenticate users, create processes, and manage teletype lines, must currently be considered within the kernel of the system, even though they are not within ring 0 but within a separate process. It is very desirable that we identify some component of this mechanism which, if properly isolated and certified, would eliminate the need to certify the remainder of the answering service. Warren Montgomery has proposed a strategy for user-authentication and process creation which would effectively remove from the kernel almost all of the current answering service mechanism. He is currently attempting to discover what problems arise from this proposed division of the function.

IX. The Use of Multiple Processes as an Organizational Tool.

As the discussion of page control suggested, an approach to understanding the structure of the system is to separate portions of it into separate processes. There are a variety of projects underway to explore this organizational structure. We are currently developing a special kind of process which runs only within ring 0, which has limited capability, and is very efficient to execute. It is our intention to use this kind of process in several applications within the system kernel. The example of page control has already been given. Another application of these processes is in the handling of interrupts. Code which executes at interrupt time is subject to several special constraints, for example, it may not abandon the processor and it may not loop on a lock. This makes code which runs at interrupt time much more complex and prone to errors. Running this code instead in a different process will eliminate these sorts of problems. Robert Mabee is currently proceeding with the implementation of this sort of process, and as a test case intends to modify the typewriter control software so that the code now running at interrupt time runs instead

in a process.

Another experiment with the use of multiple processes involve modifications of the user ring environment so that the single process of the user is conceptually shared between a number of tasks, all running in the same virtual address space. This structure, in addition to simplifying the user ring environment, seems to have several beneficial effects on the structure of the kernel. The handling of the Multics "quit" is an obvious example. The propagation of this signal through the kernel from its receipt by the interrupt handler to the ultimate process interrupt is very complex and torturous. It appears that the simplest thing for the kernel to do with a "quit" signal is to translate it immediately into a wake-up for some process. It is not clear, however, what arrangement of processes in the user ring can best take advantage of this interpretation of a "quit". The current experiment with the user ring environment will let us explore how to take advantage of this interpretation of the "quit".

X. Restructuring of Network Control Program.

Because of our group's direct involvement in the connection of Multics to the ARPA Network, we have developed significant expertise in that portion of the system. For this reason, the network is a good candidate for investigation of techniques for simplification of the system kernel. Having the network software properly structured is especially important, since it is possible that a suitable network could serve as the sole form of external I/O. For this function it seems appropriate to use multiple processes as a tool for simplification. In this case, it may be possible to remove some of the processes from the kernel thus reducing directly the bulk of the supervisor code related to the ARPA network. Notice that the ARPA network is especially interesting in this respect, since, being a multiplexed facility, some protection is required of the de-multiplexing and multiplexing function. Any technique that removes this from the kernel, without compromising its security, is an especially valuable modification to the system.

XI. System Initialization.

In order to certify the Multics system, it will be necessary to certify the "initial state" of the system; the ad hoc initialization techniques currently used makes such a certification very difficult, if not impossible. The intention of this study is to discover new ways of initializing the system which are more amenable to certification.

XII. Recovery from System Errors.

Related to the issue of system initialization is the issue of recovery from errors. They have in common the requirement that is necessary to certify or assure that the system is in some known state. We are interested in determining whether there are some particular structures for data bases and algorithms which makes it much easier to assure that the data base is in fact consistent and correct. One particular project which we are carrying out in an attempt to learn more about the structure of data bases is a comparative analysis of Multics and the Burroughs operating system, currently being prepared by Ben Williams. The Burroughs Master Control Program apparently recovers very effectively from a wide variety of errors, and we are hopeful that insight gained from an understanding of this system can help the Multics system recover from errors more gracefully and reliably.

XIII. High-Level Description of System Functionality.

As part of any attempt to certify a system, it is necessary to have some description of the intended functionality of the system itself to serve as a standard against which to certify. Several members of the project have tried various notational schemes for describing the functionality of various parts of the system. A representation of system data bases and and related algorithms in the Vienna Definition Language was performed by Richard Bratt, using the known segment table as a case study. A similar description with directory control using English as the descriptive language was performed by Douglas Hunt. Finally, Bernard Greenberg, as part of his thesis (Project MAC Technical Report TR-127), has devised a language for describing program with complexity structured data bases, which attempts to avoid implications concerning the implementation of the data base structure. This language is now being used to represent various alternative algorithms being considered as part of the restructuring of page control.

XIV. Formulation of Criteria for Inclusion of Modules Within the Kernel.

Richard Feiertag is currently attempting to develop a specific set of rules which would determine whether a module should or should not be included within the kernel of the system. He is attempting to identify these rules by studying a number of specific parts of the current system and identifying the trade offs related to moving these particular parts out of the kernel. He is currently studying the separation of policy from mechanism in page control.

XV. Study of Multics Security Holes.

An understanding of how system bugs arise and what is required to fix them gives insight into the problems of certification. For this reason we are interested in the discovery and understanding of the flaws in the current system. We attempt, periodically, to catalog all known ways to violate the security of Multics, and to identify the general class of problems of which each bug is a specific example.

XVI. Implementation of New I/O Buffering Strategy

As part of designing a simple structure for I/O, we are producing a buffer management mechanism which uses the virtual memory manager algorithm. This is described in the section of the progress report which describes CSR division work on the ARPA network.

ARPA NETWORK ACTIVITIES

This last year has been a year of transition for those working on the ARPA network, as the development effort necessary to get the network operational on Multics is gradually phasing out, to be replaced with more research oriented projects.

These research efforts on which the division is now embarking represent the best indication of the direction in which we see ourselves going in the next year.

As part of this transition, the number of staff assigned to this area has dropped from five full-time to one full-time and two half-time, with the half-time staff spending the remainder of their time working on projects related to the division's certification effort. This sharing of staff is part of our effort to unify the various efforts of the division.

The various network tasks in which the division has engaged are detailed below:

Conversion of Multics to New Hardware

Before and during the Summer of 1973, the Multics operating system was modified to run on a Honeywell 6180, rather than a Honeywell 645. It was necessary, as part of this transition, to modify the Multics software so that the ARPA network would operate with the 6180. Several sub-tasks were implied by this transition. First, it was necessary to redesign a portion of the Network Control Program so that it would interface to a full duplex rather than a half duplex network interface. Our experience, in common with that of the network community at large, was that the half duplex interface was undesirable; we seized this opportunity to convert

to full duplex. It was also necessary, as part of the move, to construct a new hardware interface between the 6180 and the IMP. The interface designed for the 645 was inappropriate, first because it was half duplex, and second because it did not support the distant interface, which we intended to use.

It has been our intention to hand off the day to day operation of the ARPA network to the staff of the Information Processing Center. For this reason, we invested some effort during the move in developing and integrating the network software so that it could be manipulated by the Multics system operators, as part of their standard procedures. This effort, which involved automating the starting, stopping, and recovering from errors of the network, was successful to the extent that we now need intervene only in very exceptional circumstances, such as a network crash or a hardware failure.

In the switching from one machine to another, the physical location of the hardware changed. For this and other reasons, the M.I.T. community obtained a second IMP. A certain amount of effort was required on our part to cut-over from the old to the new IMP. This task has been completed, and the operational responsibility for this IMP has been given to the Information Processing Center's staff.

Improvements to Network Service

During the course of the year, the Computer Systems Research Division has completed several tasks to enhance the quality of the network service provided by Multics. Several programs have been rewritten, either to enhance performance or to eliminate bugs. For example, the user interface to the Telnet protocol, which previously existed as a privately maintained program

on Multics, has been upgraded and made a part of the official Multics network software library. There is a continuing effort to enhance the mail sending facility, both in-bound and out-bound. Multics now supports a facility which allows unsendable out-bound mail to be queued for later retransmission. In-bound mail can now be delivered without the sender knowing the project ID of the recipient. We currently have designed a modification to in-bound mail handling which would decrease the cost and increase the reliability. This feature should be installed in the near future.

The network documentation has been upgraded substantially during the year. We have assembled a draft version of a Network Users' Supplement to the Multics Programmers' Manual, which gives an overview of the ARPA network on Multics, and describes how to get into Multics from the ARPA network, and how to get out to the ARPA network from Multics. In addition to the Network Users' Supplement, we are also producing a description of internal interfaces and program strategies of the system programs which support network use, to be published as one of the series of Program Logic Manuals which Honeywell is producing to describe the Multics operating system.

New Protocols

During the year our group has participated in the development and implementation of several new network protocols. Code to implement the new Telnet protocol was implemented and installed by Doug Wells. Ken Pogran has contributed to the current redesign of the file transfer protocol and plans to implement this new protocol whenever the specification has stabilized. In addition to these two protocol revisions, which have required significant effort on our part, Raj Kanodia has also proposed modifications to the host protocol which attempts to deal with lost messages in a more graceful manner than the current protocol does.

Research Topics

The division has carried out several research tasks in the network area this year; this is perhaps the most interesting work done relating to the ARPA network. In an attempt to increase the efficiency of transmitting data to and from the network, and at the same time gain a more basic understanding of the interaction between input/output functions and virtual memory computer systems, Raj Kanodia and David Clark have been devising a new buffering strategy for reading and writing from the network. The result of this design is a buffer which, by utilizing the virtual memory, appears to be infinite in length. This use of the virtual memory eliminates any need to compact or otherwise manage the buffer area, thus reducing overhead. The buffer is also directly accessible to a user process, since it is in the virtual address space; this avoids the necessity of copying data in order to make it accessible. It is our intention to attempt to use this same buffering strategy to interface the typewriter control processor as well as the ARPA network; it appears that this strategy can be exploited successfully for all devices for which the system's nucleus must take responsibility. This unification of buffer management, by reducing the bulk and complexity of the kernel, is a significant contribution to our certification project discussed earlier in the report.

As part of our research into the use of the ARPA network as a vehicle for communication between active processes, Warren Montgomery and Ken Pogran have been studying and implementing the RSEXEC protocol devised by Robert H. Thomas of Bolt, Beranek and Newman. Initial investigation of this protocol suggests that parts of it can be adapted easily for our operating system, but the other parts, in particular the manipulation of files, are difficult to dovetail into our particular view of a storage system and user interface

thereto. More research is intended in this area, as manpower permits; we hope that a partial implementation of RSEXEC will become available on Multics.

In a project related to the previous one, we have currently established a process in Multics which is always available for the purpose of providing services related to the ARPA network. Currently, this process will queue and retransmit mail which for some reason is currently undeliverable across the network. In the near future, we intend that this process will handle in-bound mail, and support those functions of RSEXEC which we are able to implement.

As part of our redesign of the buffering strategy for the network, Art Benjamin has attempted to increase the precision of the meters which tell us about traffic flow between Multics and the ARPA network.

Michael Padlipsky has developed the specification for a network-wide text editor, called neted, which has been implemented and used at eight different sites on the ARPA network. It is felt that this editor serves as a small but valid example of the fashion in which one standard command interface can be adapted to various time sharing systems, with their various user interfaces.

Network Committees

During the year our group has contributed to the network effort by active participation in several ARPA network committees. These include the "USING" subcommittees which are specifying the common command language and neted; the committee which is redesigning the file transfer protocol; the Review Committee for the ARPA network terminal

system (ANTS) and the ELF operating system for PDP-11; the Graphics protocol committee; and an informal group redesigning the ARPA network host-to-host protocols.

TECHNOLOGY TRANSFER

The term "technology transfer" may be loosely applied to making research results accessible to industry, government, and education institutions. Since, in undertaking the Multics project, the Computer Systems Research Division developed a sizable store of technology, transfer of that technology has recently been a major activity. Several specific points indicate the progress of this transfer:

1) In April, 1974, Honeywell Information Systems Inc. announced availability of an upgraded hardware system for Multics, the model 68/80, which includes a cache memory in each processor and support for a large (2-8 million words) directly addressable memory.

2) Honeywell 6180 Multics installations were completed at M.I.T., the Air Force Data Services Center, Ford Motor Co., and General Motors. These installations are in addition to the older Honeywell 645 installation at Rome Air Development Center and also Honeywell internal Multics sites in Waltham and Billerica, Mass., Phoenix, Arizona, Paris (Honeywell Bull) and Tokyo (Toshiba). In total, as of June 30, 1974, there were three 645 sites and seven 6180 sites running Multics. Each of the non-Honeywell sites has initiated contact with M.I.T. to facilitate more direct transfer of knowhow and, in some cases, of M.I.T.-developed Multics subsystems.

3) The M.I T. Information Processing Center and the Telefunken Co. completed negotiations for Telefunken's acquisition of knowhow and rights to utilize the Multics PL/I compiler.

4) As development of the Multics ARPANET attachment has matured, transfer of that technology to Honeywell has begun. Although not yet fully supported by Honeywell, the network attachment is currently available to other Honeywell customers as an option, using M.I.T.-supplied software.

5) One M.I.T. staff member versed in ARPANET development has moved to the MITRE Corporation to help implement a government network similar to the ARPANET.

6) Of the ten undergraduate and graduate students of CSR who completed their educational programs, four accepted positions with Honeywell.

7) Faculty members of the Computer Systems Research Division have been actively consulting with several different government and industrial organizations; these consulting activities largely consist of translating the Multics experience into forms applicable to other system designs. Organizations include Control Data Corporation, IBM Corporation, General Telephone and Electronics, and System Development Corporation, and groups within the Department of Defense. In a related activity, the Computer Systems Research Division entertained several industrial visitors interested in Multics technology, many making contact through the M.I.T. Industrial Liason Office.

8) The first draft of a tutorial paper on the protection of information in computer systems was completed by Professors Saltzer and Schroeder. This paper captures for educational purposes many of the insights discovered in the course of Multics development. Also made available in Project MAC Technical Report form was the introduction to the Multics Programmers' Manual.

9) Japanese interest in computer systems technology has been high for several years; in most recent periods we have had one or more Japanese visitors. This year, Professor Eiiti Wada of the University of Tokyo joined the division as a visiting Associate Professor. Also, two books about Multics were published (in Japanese) by former visitors. The first was a translation of Elliott Organick's book "The Multics System" by Akio Sasaki and Toyohiko Kikuchi. The second, a new book entitled "Structure of Computer Utility - Anatomy of MULTICS" was written by Professor Katsuo Ikeda of Kyoto University.

OTHER ACTIVITIES

During the current reporting period Masters' theses by Robert M. Frankston and Lee J. Scheffler have been completed under the supervision of F. J. Corbató

In the thesis by Frankston, entitled "The Computer Utility as a Marketplace for Computer Services," the implications of widespread commercial use of a computer utility were explored. As stated in the abstract:

> "Most contemporary computer systems are oriented towards users who run programs. The environment for services puts different requirements on the computer system than do the needs of programmers, so as to permit all the participants in the market to make effective use of its facilities without requiring dedicated facilities and without interfering with each other. As with any marketplace, it must be convenient to do business within its framework."

The thesis, available as MAC TR-128, also includes as an example a design evolved from the Multics System which implements a particular marketplace model.

"Performance Evaluation of Rotating Disk Subsystems in Multiprogrammed Computer Systems" is the title of the thesis by Scheffler. This thesis analytically derives, using primarily the mathematics of Operations Research,

expressions (or in some cases bounds) for the performance of various common classes of disk subsystems subject to work loads typical of large-scale multiprogrammed computer utilities. As stated in the abstract, this study

> "... is fundamental to 1) predicting the overall performance of the system; 2) enhancing disk subsystem performance to meet evolving secondary memory performance requirements; and 3) choosing disk subsystems to meet stated performance requirements."

The derived expressions were validated against measurements taken on the Multics System at M.I.T.

Another Master's thesis, by Bernard Greenberg, developed a set of measuring tools for exploring the paging rate of Multics for very large memory sizes. This work involved modifying Multics to intercept all movements of data to and from the disk and thereby record an address reference string. The reference string is recorded during actual Multics operation, and later analyzed with a least-recently-used algorithm simulator to discover what disk traffic rate would have resulted with larger main memory sizes. The measurement is complicated by a variety of real-life factors such as Multics cleverly not writing out pages which happen to contain all zeroes, and Greenberg succeeded in providing suitable corrective measures for each of them. Unfortunately the development and proof that the measuring tool worked was a thesis in itself; a systematic program of measurements with the tool awaits some other student.

A Bachelor's thesis, by Gordon Benedict, developed two software implementations of the IBM "LUCIFER" enciphering algorithm. The purposes of this activity were three fold:

1) to discover whether or not the IBM disclosure of the LUCIFER system was actually complete enough to permit its duplication,

2) to establish one example of the potential performance of software encryption strategies on the Honeywell 6180 computer,

3) to provide a tool with which to experiment with several proposed information protection strategies such as end-to-end encryption of ARPANET links, and encryption of files stored on line.

The encryption study is of interest both to the ARPANET activities and also to the information protection and certification study.

PUBLICATIONS, TALKS AND THESES

Publications

Ikeda, K., Structure of Computer Utility - Anatomy of MULTICS, Shokodo Co., Ltd., Tokyo, 1974.

Multics Programmers' Manual, Part I (Introduction), Revision 14, September 30, 1973.

Multics Programmers' Manual, Part II (Reference Guide), Revision 15, November 30, 1973.

Saltzer, J. H., "A Simple Linear Model of Demand Paging Performance," Comm. ACM 17, 4 (April, 1974), pp. 181-186.

Schroeder, M. D., "Proceedings of the ACM SIGPLAN/SIGOPS Interface Meeting," SIGPLAN Notices 8, 9 (September, 1973), (editor with R. M. Graham).

Schroeder, M. D., "A Brief Report on the SIGPLAN/SIGOPS Interface Meeting," Operating Systems Review 7, 3 (July, 1973), pp. 4-9.

Technical Reports Published

Rotenberg, L. J., "Making Computers Keep Secrets," Project MAC Technical Report TR-115 (Ph.D. thesis, M.I.T. Department of Electrical Engineering, June, 1973).

Stern, J. A., "Backup and Recovery of On-Line Information in a Computer Utility," Project MAC Technical Report TR-116 (E.E. thesis, M.I.T. Department of Electrical Engineering, August, 1973).

Clark, D. D., "An Input/Output Architecture for Virtual Memory Computer Systems," Project MAC Technical Report TR-117 (Ph.D. thesis, M.I.T. Department of Electrical Engineering, August, 1973).

"Introduction to Multics," Project MAC Technical Report TR-123, November, 1973.

Greenberg, B. S., "An Experimental Analysis of Program Reference Patterns in the Multics Virtual Memory," Project MAC Technical Report TR-127 (S.M. thesis, M.I.T. Department of Electrical Engineering, January, 1974).

Frankston, R. M., "The Computer Utility as a Marketplace for Computer Services," Project MAC Technical Report TR-128 (E.E. thesis, M.I.T. Department of Electrical Engineering, January, 1974).

Janson, P. A., "Removing the Dynamic Linker from the Security Kernel of a Computing Utility," Project MAC Technical Report TR-132 (S.M. thesis, M.I.T. Department of Electrical Engineering, May, 1974).

Other Theses Completed

Gumpertz, R. H., "The Design and Fabrication of an ARPA Network Interface," S.B. thesis, M.I.T. Department of Electrical Engineering, July, 1973.

Scheffler, L. J., "Performance Evaluation of Rotating Disk Subsystems in Multiprogrammed Computer Systems," E.E. thesis, M.I.T. Department of Electrical Engineering, June, 1974.

Benedict, G. G., "An Enciphering Module for Multics," Project MAC Technical Memoradum TM-50 (S.B. thesis, M.I.T. Department of Electrical Engineering, June, 1974).

Theses Completed, continued

Goulet, L. J., "A Dynamic Peripheral Allocation Scheme Between Identical Processors," S.B. thesis, M.I.T. Department of Electrical Engineering, May, 1974.

Vogel, D. A., "Manufacturing Technical Data Systems," S.B. thesis, M.I.T. Department of Electrical Engineering, June, 1974.

Theses in Progress

Bratt, R. G., "Removing Name Space Management from the Security Kernel of a Computer Utility," S.M. thesis, M.I.T. Department of Electrical Engineering, expected date of completion, December, 1974.

Reed, D. P., "A Simple Implementation of Processes as a Basis for a Certifiable Computer Utility," S.M. thesis, M.I.T. Department of Electrical Engineering, expected date of completion, January, 1975.

Bishop, P. B., "Very Large Address Space Modularly Extensible Computer Systems," Ph.D. thesis, M.I.T. Department of Electrical Engineering, expected date of completion, June, 1975.

Talks and Presentations

Saltzer, J. H., "Engineering Insights Based on Experience with Multics," series of six lectures given at the Technion International Summer School on Time-Sharing, Haifa, Israel, July 8-10, 1973.

Saltzer, J. H., "Protection and the Control of Information Sharing in Multics," presented at the Fourth ACM Symposium on Operating System Principles, Yorktown Heights, New York, October 15-17, 1973.

Thomas, E. L., "Proposed ARPA Network Graphics Protocol," presented at the Workshop on Machine-Independent Graphics, National Bureau of Standards, Gaithersburg, Maryland, April 22-23, 1974.

Pogran, K. T., "Using Multics ARPA Network Software," presented at the University of Illinois Center for Advanced Computation, October 25-26, 1973.

ARPA Network Committee memberships

Padlipsky, M. A.: User Interest Group (USING) and subcommittees.

Thomas, E. L.: ARPA Network Graphics Group

Pogran, K. T.: ARPA Network Graphics Group,
ARPA Network Terminal System Steering Committee,
File Transfer Protocol Committee.